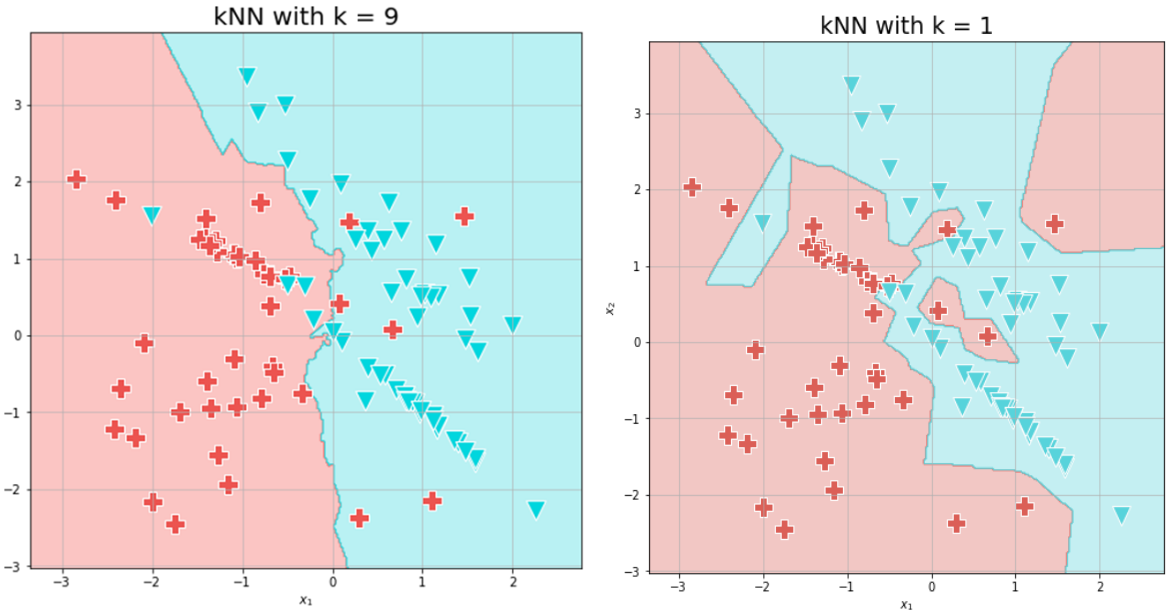
Submitted by: Eden Dembinsky 212227888 & Assaf Lovton 2098444414 - HW2

Part 1: k-Nearest Neighbors (model implementation and selection):

Q1.



For k=1, we can see that we get an overfitting,large variance since we take into consideration only the closest sample.
We can see that even around a very dense area of blue samples around the center (0,0) we get a classification of red even though the reasonable classification is blue
Another occurrence is around (0,1.5) area where we get a big area classified as red instead of blue although we can see a lot of blue samples around this area.

For k=9, we can see that we get an underfitting, large bias since we take into consideration 9 neighbors and thus a lot of far points affect the point predicted label.

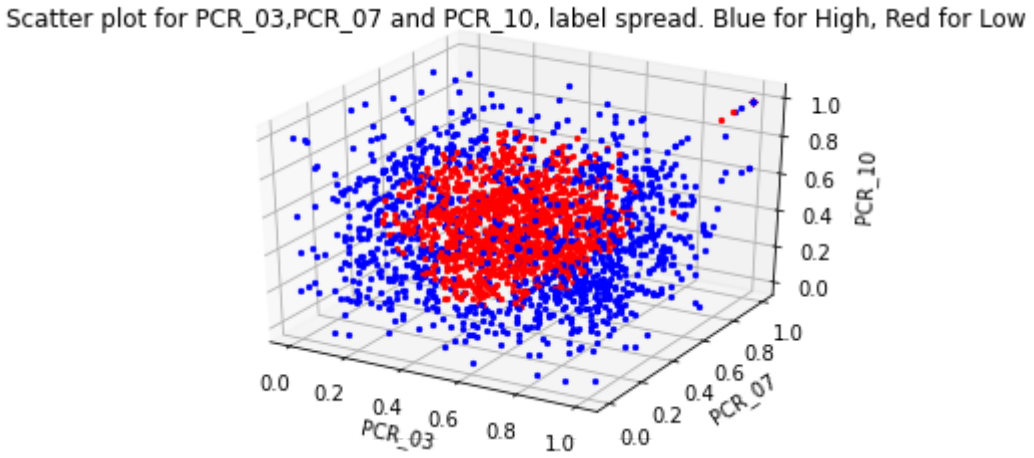
We can see that using a large neighborhood we get a large bias, inaccurate estimation while with a low neighborhood we get an uninformative model which is noisy (large variance).

Q2.

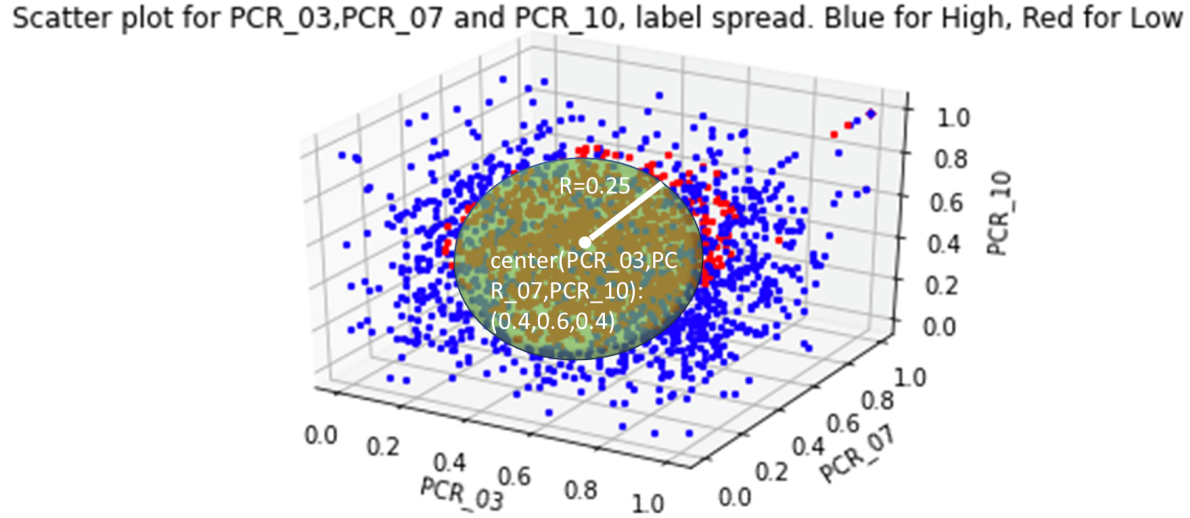
PCR_10 0.151594
blood_type_A+ 0.059768
PCR_07 0.046893
household_income 0.034036
sex 0.028666
symptoms_shortness_of_breath 0.023319
sugar_levels 0.022069

symptoms_cough	0.018483
PCR_01	0.016312
PCR_06	0.013586

Q3.



Q4.



As we can see there is a shape of a ball.
We will predict Low spread for PCR_03, PCR_07, PCR_10 s.t:
(PCR_03 - 0.4) ^ 2 + (PCR_07 - 0.6) ^ 2 + (PCR_10 - 0.4) < 0.25^2
Otherwise, we will predict High spread.
That means we predict low for samples inside the ball we defined above, and High for samples outside the ball.

Q5.

The training accuracy is: 0.87875.

Data Normalization:

Q6.

Z-Score is When you apply the technique of scaling the features in such a way that they end up having properties of a standard normal distribution with mean equal to zero and standard deviation of one.

MinMax Scaling is when you scale the features in such a way that the values usually range between -1 to 1

The min-max scaling results in smaller standard deviations than the Z-Score this will **decrease our ability to deal with outliers**.

Z-Score can result with a feature that is not between -1 to 1 in contrast to minMax that is certainly between -1 to 1.

When the feature is distributed normally we will choose to normalize with the Z-Score.

Q7.

We got 0.9114 using the Z-score normalization and we got 0.9225
using the min-max normalization. We can see that using both normalization options we got an improvement of 9 percent. We can see that before the normalization PCR_03 had a range of zero to 350 while PCR_10 that had a range of 0 to 8, that means that PCR_03 had a bigger impact on the classification results since we used euclidean distancing to calculate the distance between the neighbors. Now after we normalized the data each feature had a proportional impact on the result, causing a much better accuresis.

Q8.

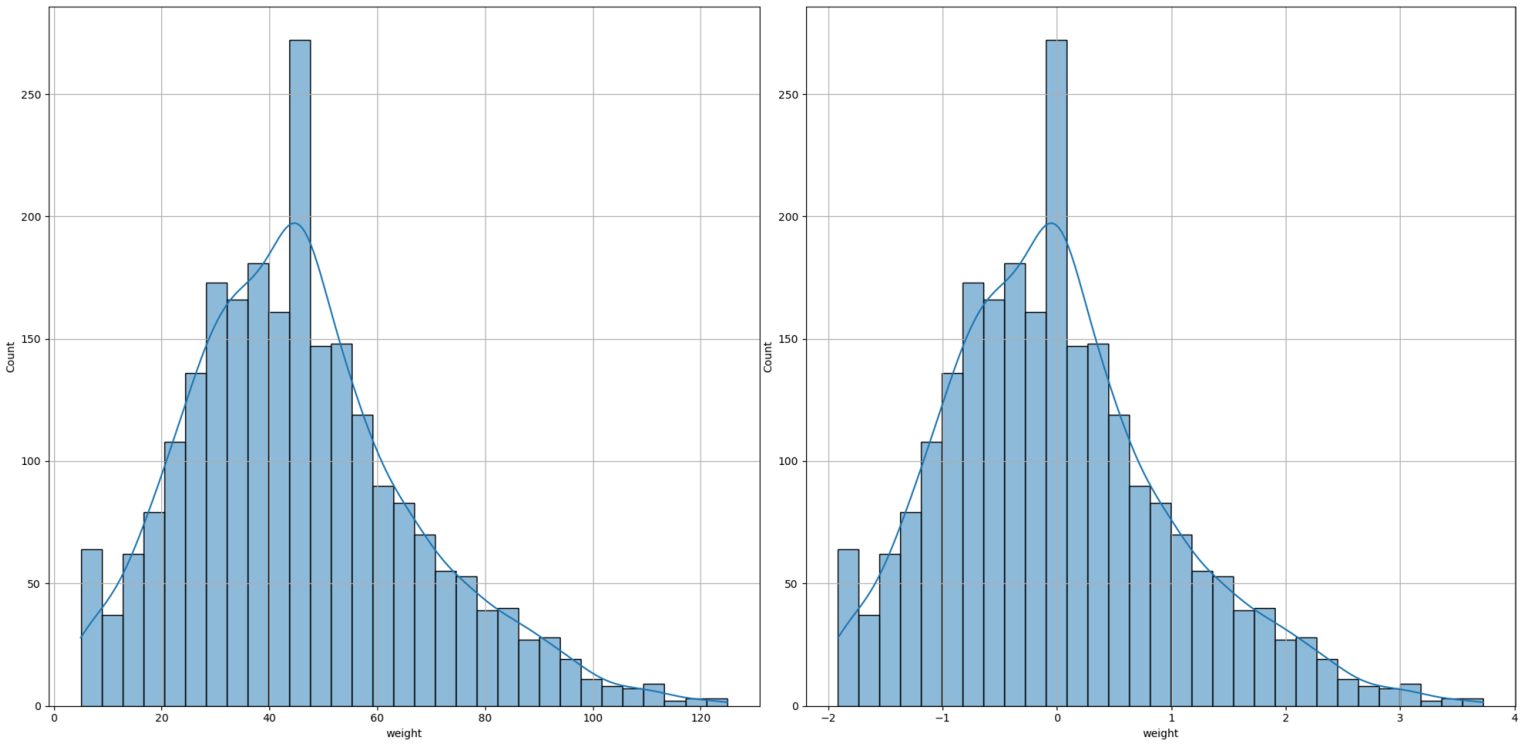
We choose the weight and the number_of_sibilings.

For the weight feature we choose to normalize with Z-Score because we believe that this feature is normally distributed and also in these feature the outliers are important to consider because a person with a specially large weight may have more risk to have covid, as we explained before min-max scaling reduces the effects of outliers.

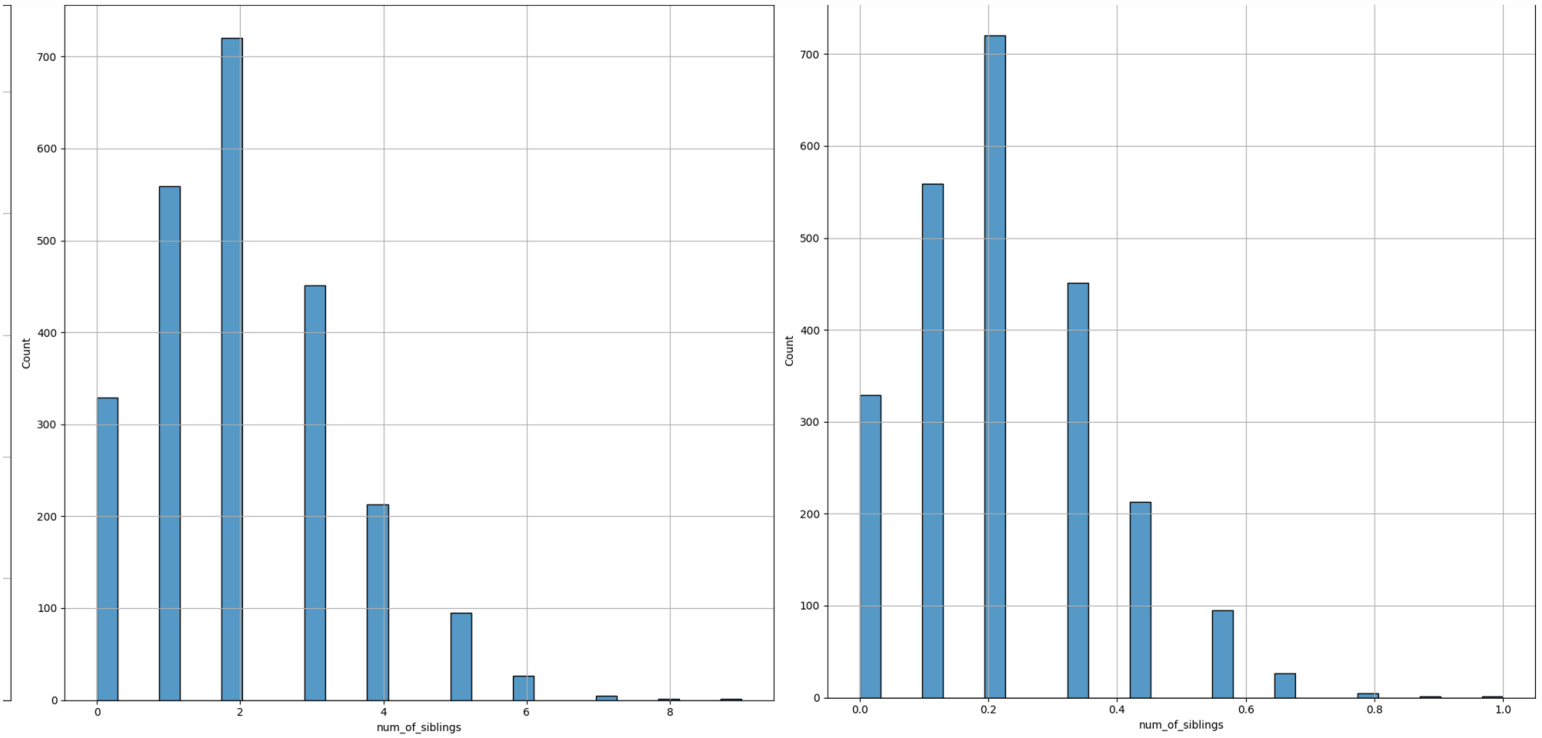
For the number_of_sibilings feature we choose to normalize with min-max scaling.

We think that for this feature outliers are not that important so we can use min-max scaling to make this feature value between -1 to 1 and maybe benefit from that in our model results. Z-Score does not guarantee values between -1 to 1.

Histogram of weight before (left) and after (right) Z-score normalization



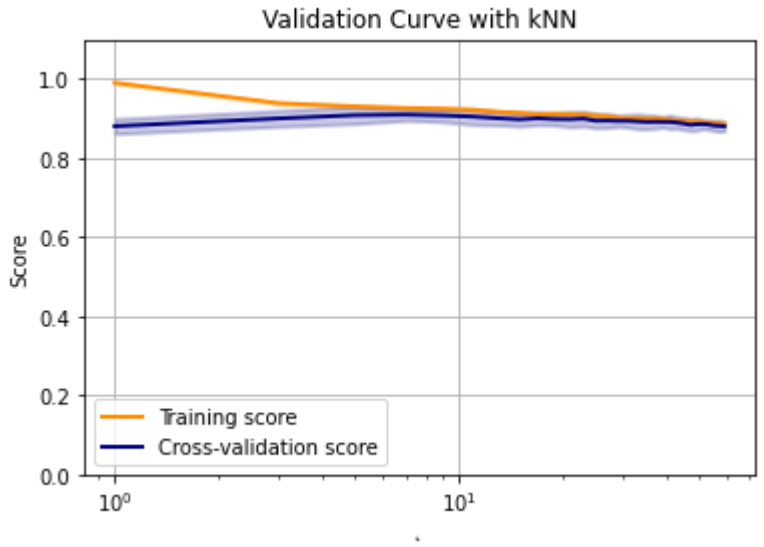
Histogram of num_of_siblings before (left) and after (right) min-max normalization



Model selection (hyperparameter tuning):

Q9.

As we can see we got that for k = 7 we got the best results for the cross-validation accuracy, max of 0.9095833333333333, and for the training 0.9134.
The max of the training score is for k=1 and obviously score of 1.



Error analysis:

Q10.

Our model tends to make more False Negative mistakes.

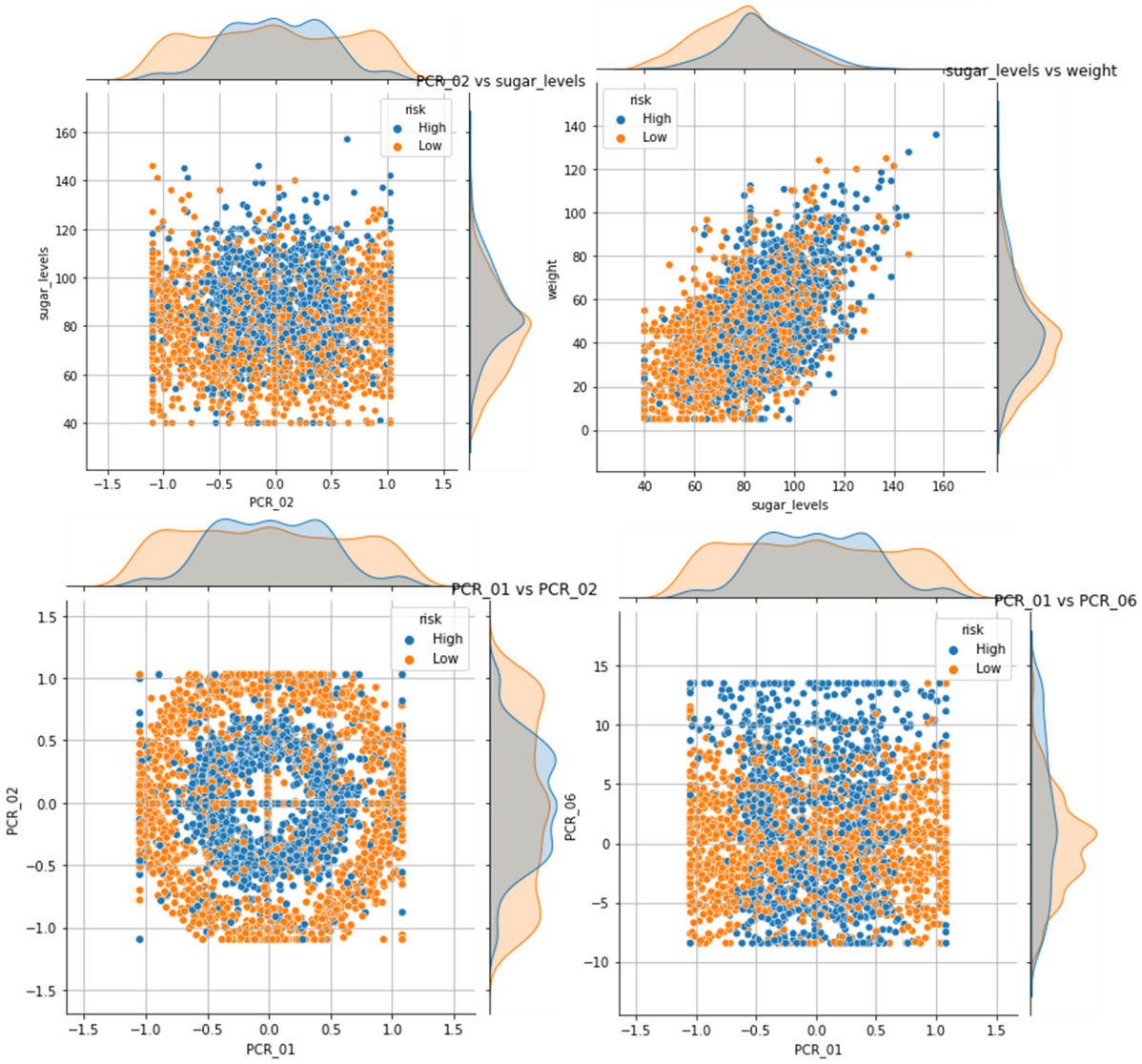
	label positive	**label negative**
predicted positive	1051	142
predicted negative	145	1062

Part 2: Decision trees (model visualization and selection):

Visualization and basic analysis:

Q11.

As we can see PCR_01, PCR_02, and suger_levels are the three features that seem the most promising for prediction.



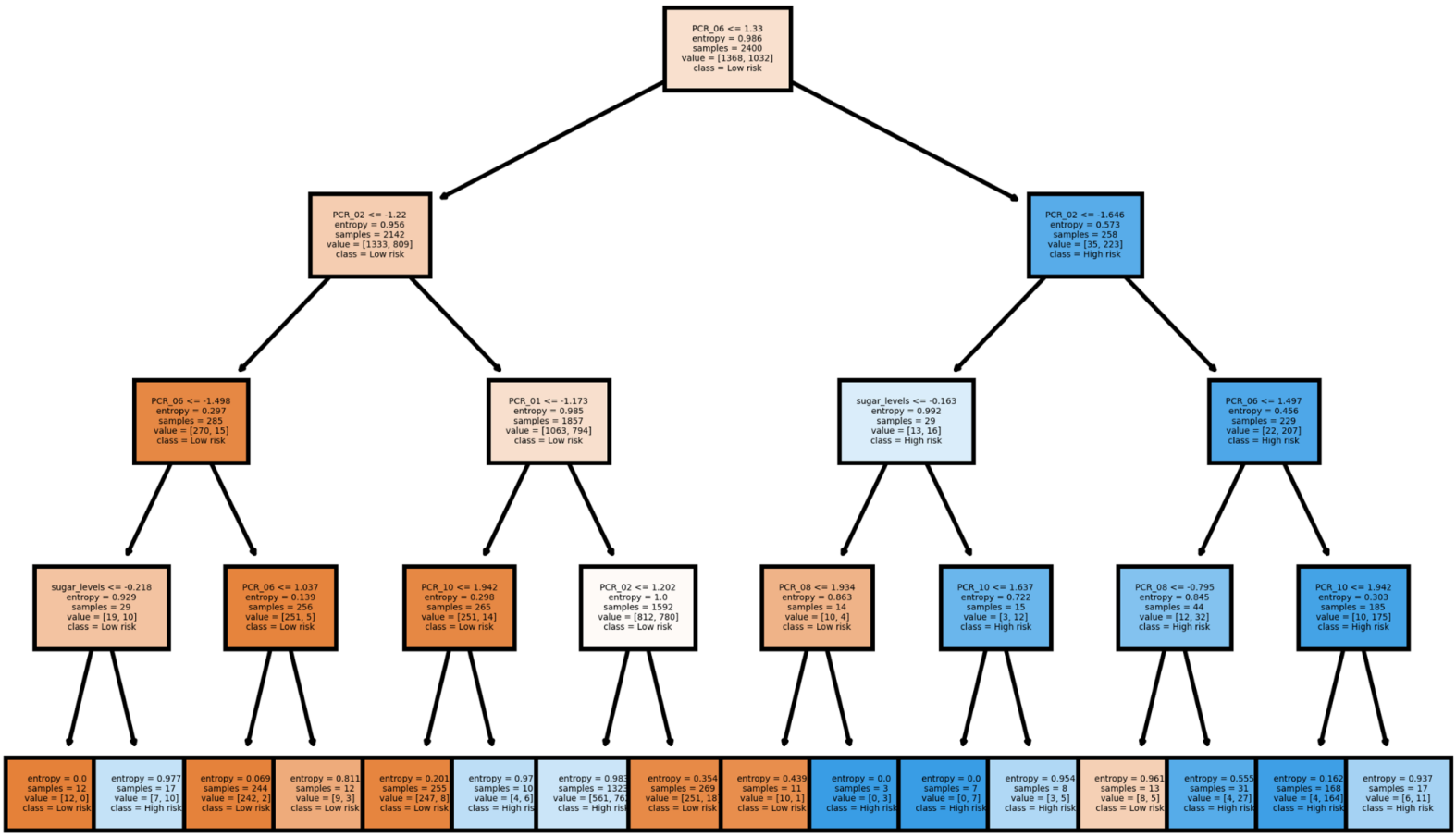
Q12.

sugar_levels	0.262572
PCR_06	0.194540
weight	0.158122
PCR_03	0.037509
blood_type_A+	0.032901
symptoms_fever	0.027266
PCR_02	0.026809
num_of_siblings	0.022001
sex	0.019875
PCR_01	0.019871

Q13.

Model accuracy score with criterion entropy: 0.7392

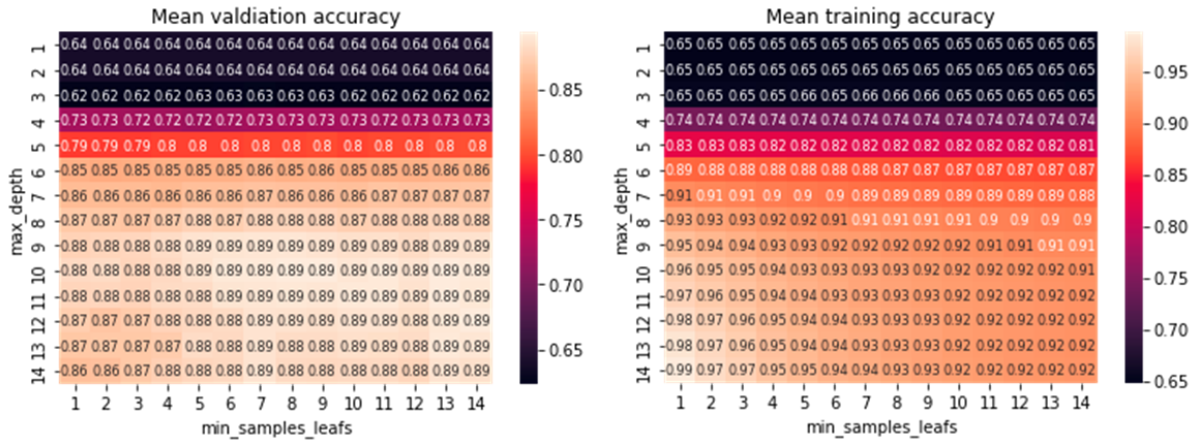
Decision tree with ID3 on PCR_01,PCR_02, PCR_06 & sugar_levels predicting risk target feature



Q14.
We did choose a feature that has high correlation, sugar _levels, but we also took features from the lower bottom (PCR_01 and PCR_02).
We chose PCR_01,PCR_02 based on a deeper analysis - pairplotting, that added a new dimension to the data, we were able to see how a combination of two features separates the data. Also we can see that we made good choices since the ID3 did use all of the features we have selected, if we would have asked to provide 4 features we would have also included PCR_06 as you can see already in Q11's graphs we provided the pairplot showing it was also a good feature to estimate risk.
So although PCR_01, PCR_02 have low correlation with the risk target feature, they were important for classifying risk, as we can see in the pair plots and the tree above they provided a clear separation of the data.

Model selection:

Q15.
(c). If we look at the mean validation accuracy we can see that the highest accuracy we can get is 89, yet we would like to do that with minimal max_depth since it will result in less calculation in our model, moreover we would like to have large enough min_samples_leafs. We can see in the graph that the smaller and lower we are get better training accuracy but if we go too much we start hurting our validation accuracy, so we want some parameters to balance each other.
Regarding all that said above we found max_depth = 9 and min_samples_leafs = 7 best at this task.



For those optimal parameters we get Score of:
validation accuracy : 0.89375
training accuracy: 0.9363

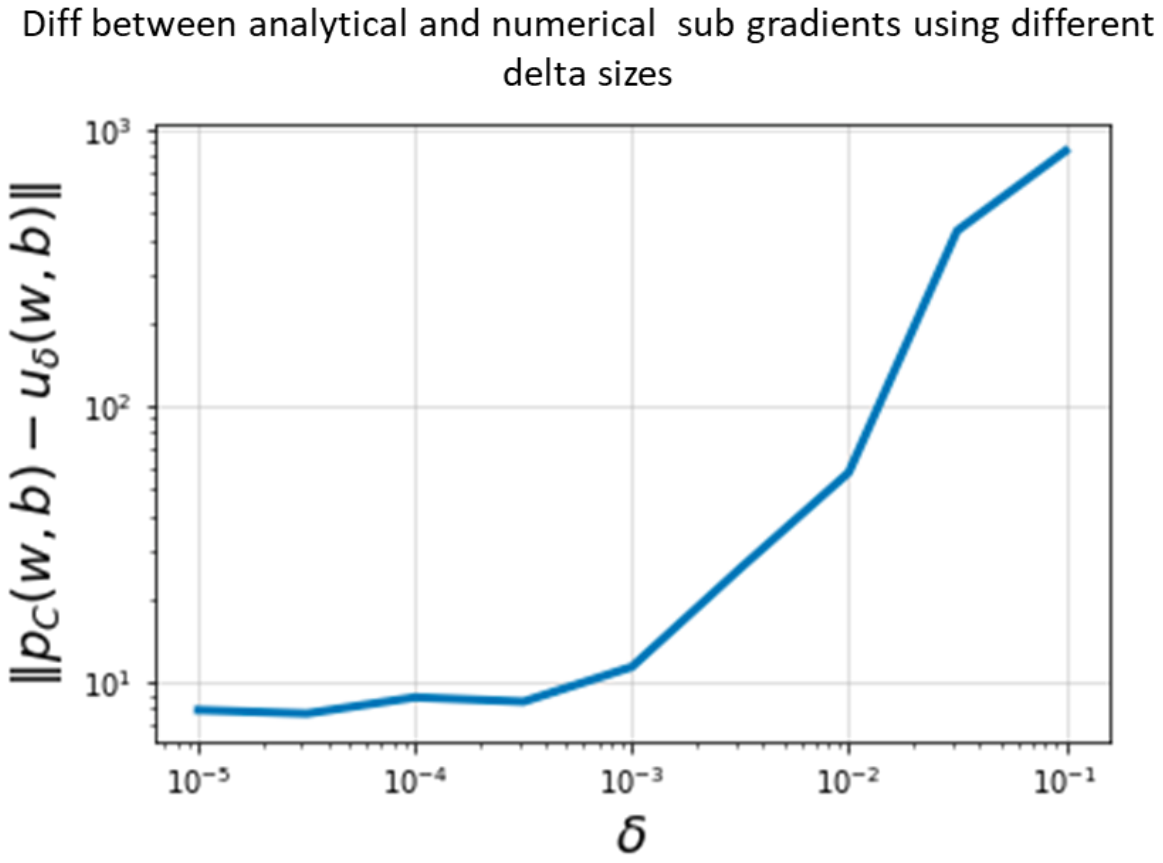
(d). For underfitting we will take max_depth = 1 and min_samples_leafs = 14. We get a training accuracy score of 0.65 and validation accuracy score of 0.64, both scores are very low due to the low max_depth of the graph. Obviously with only one level in the tree we cannot lower the entropy enough to make a wise decision.

(e). For overfitting we will take max_depth = 14, and min_samples_leafs = 1. For the training we get an accuracy score of 99 while for the validation we get only 0.86, we can definitely

see that we passed the sweet spot of the fitting and we fit the training data in a way that damages the performance of our model.

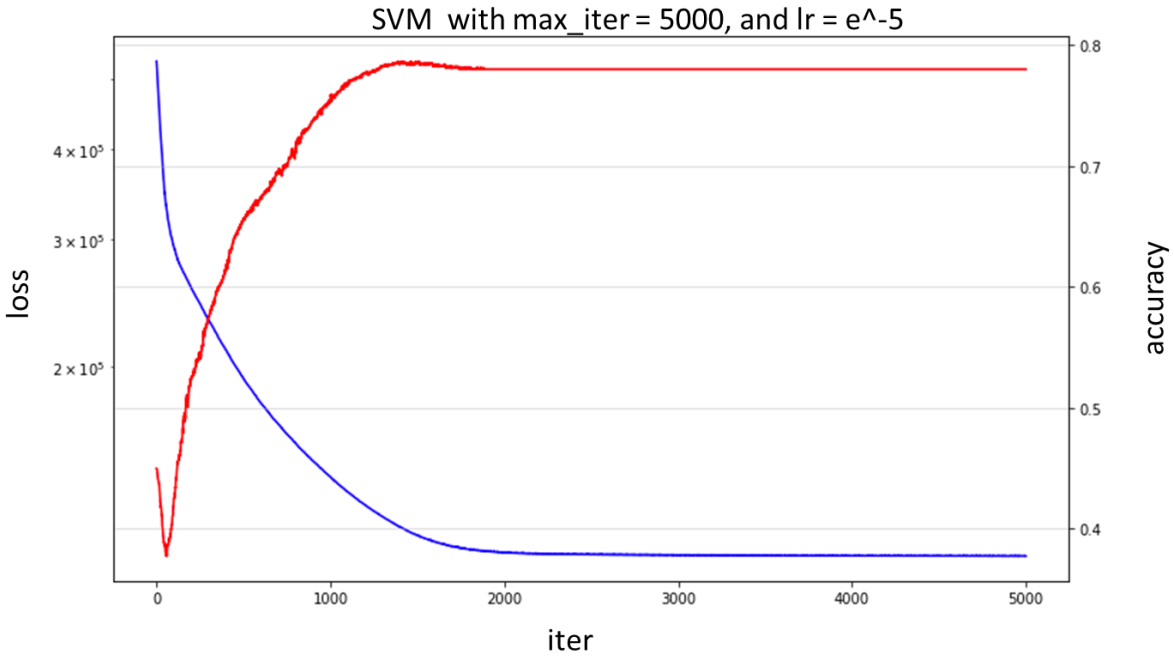
Part 3: SVM (implementation and optimization):

Q16.
As we can see in the graph, for bigger delta we get a bigger norm.
As the delta grows the distance between the numerical and analytical subgradients grows as well. That makes sense since by the definition of derivative (or taylor series) we get better approximations to the derivative when the delta is small.

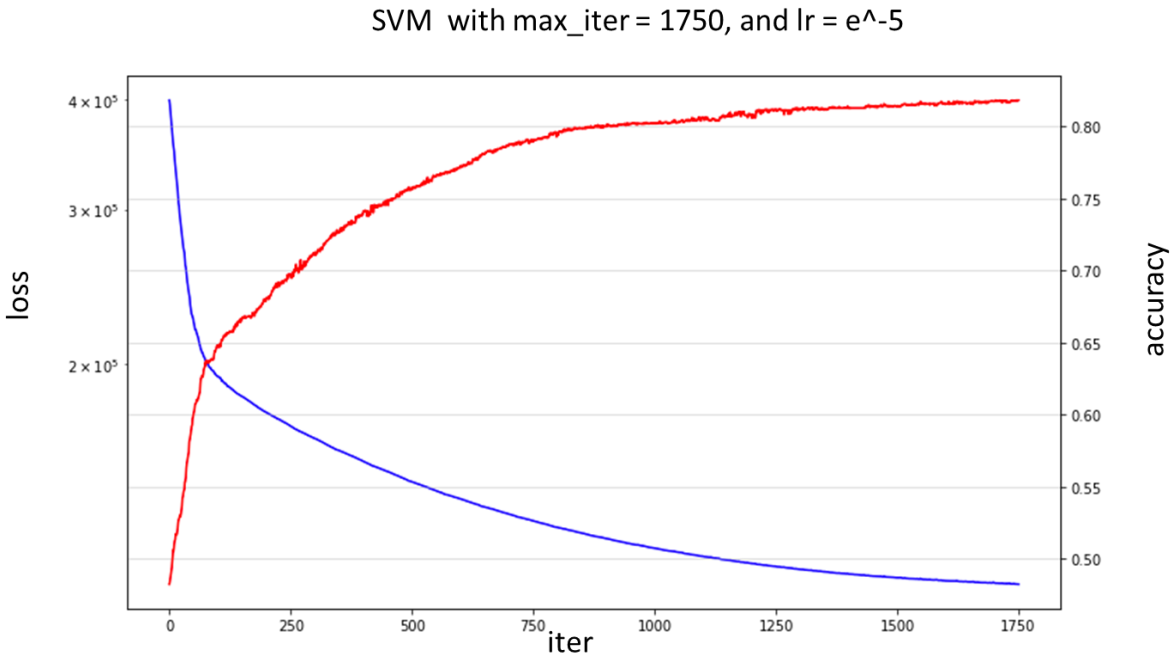


Solving Soft SVM problems using Stochastic Gradient Descent (SGD:

Q17.
We tried many different combinations, the best result we got was using lr = 1*10^-5 and yet we saw that accuracy already converges in iter =1750 .



As we can see we get to the maximum on iter 1500 so we adjust our max_iter to apply early stopping, so we decided max_iter = 1750.

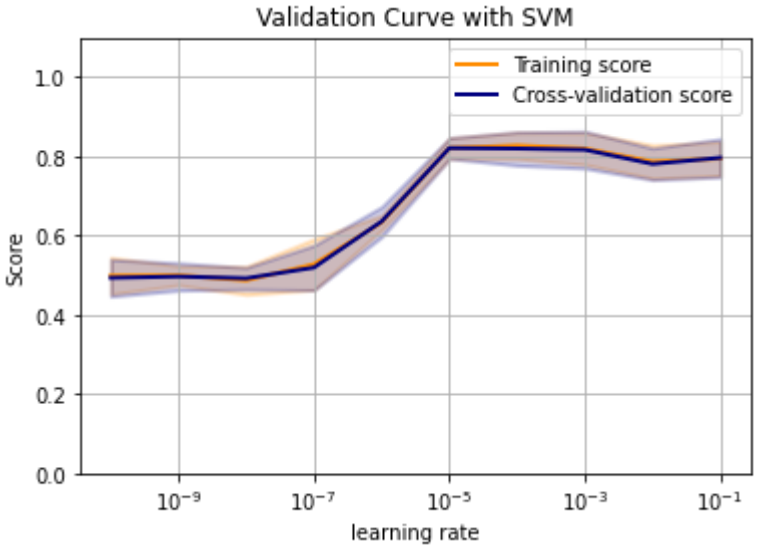


Applying a feature mapping:

Q18.
As we know SoftSVM uses linear hyperplanes to separate classes, while
KNN can generate a highly convoluted decision boundary as it is driven by the training data itself. In the analysis we made with the PCR data from question 7, we saw that the hyperplane that is created by PCR_03,PCR_07,PCR10 is definitely not linearly separable when looking at the target feature spread. kNN isn't restricted to linear separation thus, it's still doing a good job because the classes are almost sperable as a ball around the red points, as we explained before.

Q19.
As we can see the best results we got were: best_lr = 10^-5, best_acc = 0.823749 .

We already saw and expliend in Q18 that the data can be separated with a Spherical plain, thus we will be able to use our model if we apply transformation on our data. For learning rates under 10^-5 we can see that we achieve lower scores that's because the learning rate is too small which leads to slower convergence to the minimum point,(every iteration we go in the opposite direction of the gradient but the step is limited by the learning rate) as we can see we don't even get close to the miniumon under 2000 iteration, we might have been able to get there with more itratiations (slow and steady). Moreover we can see that for learning rate bigger than 10^-5 we start to get lower accuracies that's due to larger steps causing us to jump over the minimum point (although we don't go too far from there, if we would have increased the learning rate we would have decreased in our accuracy score dramatically). Also we can see that for every learning rate the training score accuracy is very close to the cross validation accuracy that implies that our algorithm generalized very well.



Q20.
The kernel SVM exploit that fact the the dual SVM problem only depends on the inner product of xi with xj, thus we need to calculate only the inner product of the transformed xi,xj

and we don't need to save all of the transform x, this fact is crucial because for polynomial for example the number of features created growing exponentially.
While in question 19 the polynomialFeatures function creates a full matrix of the high dimensional features, for degree 2 this was mangable but for higher dimension this will cause a problem.

Q21.
As we know kNN predicts by memorizing all the data, i.e the algorytm saves in the memory all of the data in the fit function, (n_fetures*n_samples) = (3*2400) = (7200).

The SoftSVM model only saves: weights vector, bias, C constant, max_iter constant, batch_size constant.
The weight vector size is the number of features and others are negligible.
additionally the 2 degree polynomial feature saves 3^2 features, i.e 9 features.
So, the memory for the SoftSVM model is around (n_features_mapping) = 9.
This is a huge difference that explains the difference in the accuracy of both models.
That’s the tradeoff between accuracy and memory.