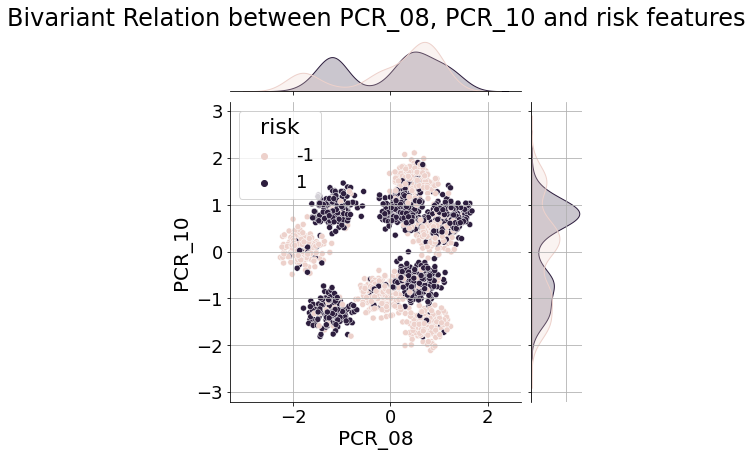
**Major HW 2 - Algorithm Implementation and Basic Model Selection**

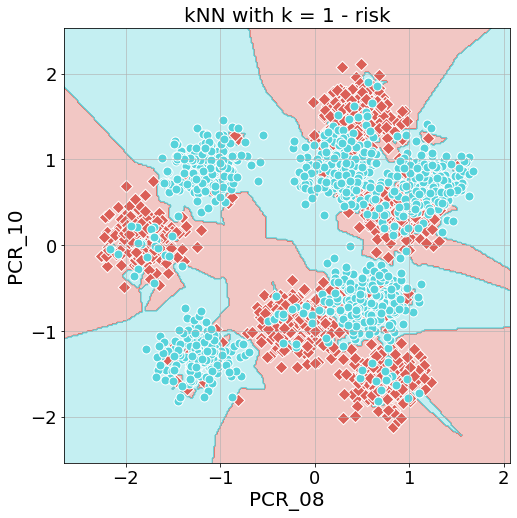
**Naama Shamgar: 204893952**

**Liran Cohen: 209043470**

1. Below is the joinplot describing the correlation between PCR\_08, PCR\_10 and the risk target variable:



1. Below is the visualization of the decision regions of a trained knn model with k=1, for PCR\_08, PCR\_10 to predict the risk target variable:



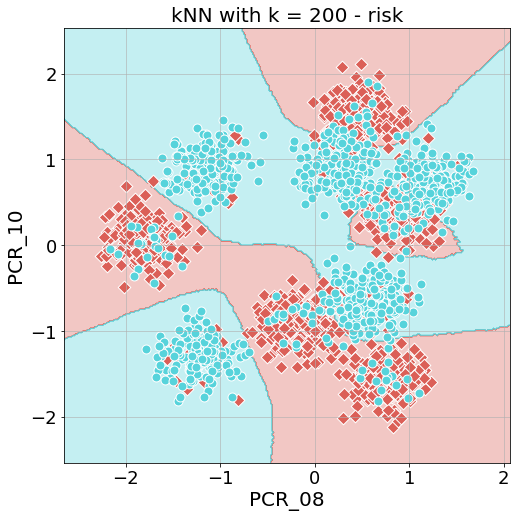
1. Below is the validation curve displaying mean train accuracy and mean validation accuracy for different k values:



The best value for k, i.e. the value for which the validation accuracy is the highest (or the validation error is the lowest) is k = 200. Its mean training accuracy is 0.864, and its mean validation accuracy is 0.8625.

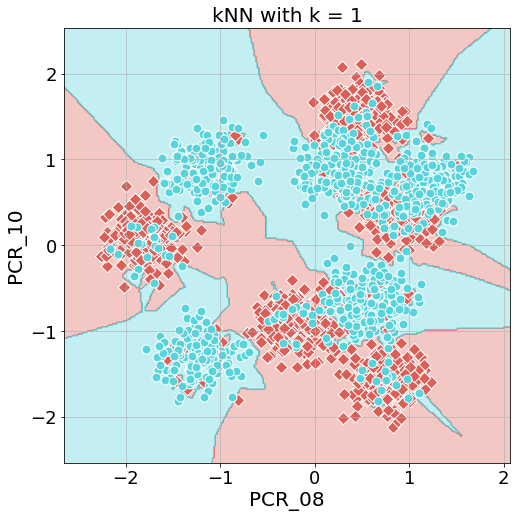
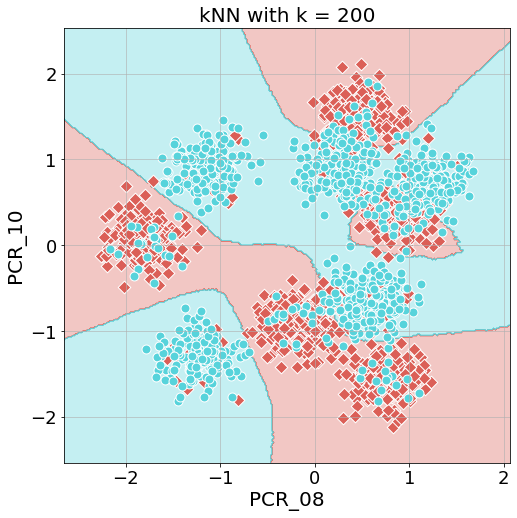
We can see from the plot that for lower values of k (approximately 5 and under) the training accuracy is higher than the validation accuracy. These values indicate overfitting, which is logical since we’re only considering a very small number of neighbors, causing the predictions of the model to be specific to the training samples we use every time. On the other hand, we can see that for higher values of k (approximately over 300) the training and validation accuracy drop, indicating underfitting. This is caused by considering too many neighbors for every sample (even neighbors that show no similarity to the predicted sample), making the results general and non specific.

1. Below are the decision regions of the kNN model with k = 200 (our optimal value), trained on all training samples:



The test accuracy of this model is 0.85 (calculated on the test samples).

1. We will now compare the kNN models with k = 1 and k = 200 side by side:



We can see that the decision regions are somewhat similar, but the boundaries for k = 200 are much smoother and less noisy. This fits our earlier assumption that lower values of k would overfit - almost each individual point causes a new region for k = 1 (noise), making the decision regions fitted exclusively to the specific data we are using to train the model. This behavior is very different from the behavior of the k = 200 model - only taking into account significant groups of samples with the same label, and ignoring stragglers.

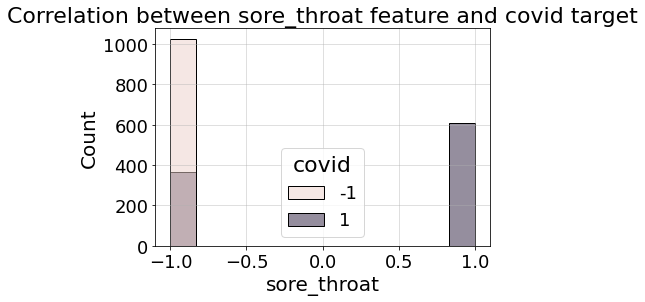
1. Below is the validation curve displaying mean train accuracy and mean validation accuracy for different k values, using all the features in our dataset:



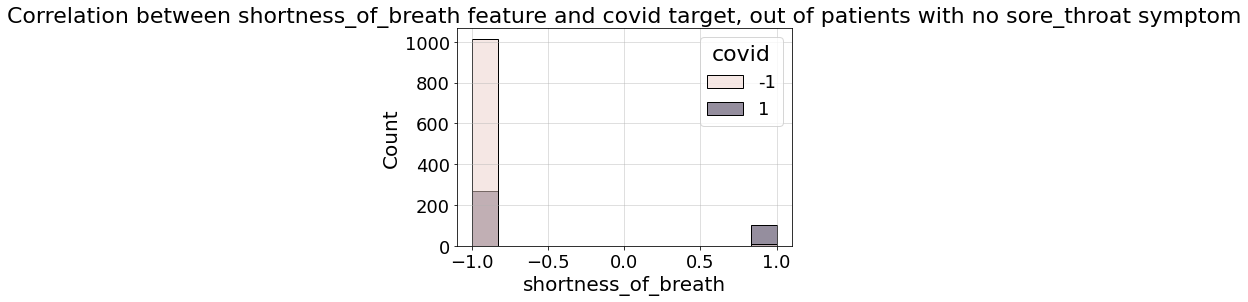
As we can see in the validation curve, The validation accuracy is consistently low for all values of k (approximately 0.6). The reason for this is the number of features we used - 19. This means that the distance measure kNN uses needs to consider 19 dimensions with the same amount of samples it had for 2 dimensions. Because of this reason, the accuracy of the model dropped significantly - the distance measure it needed to compute was too complex for the amount of data we have (the curse of dimensionality). Additionally, we used many features that showed no correlation to risk, meaning the model took into consideration essentially random values, which lead to low predicting accuracy. For the training accuracy, we see higher scores for lower values of k (again we see overfitting), but it too reaches lower accuracy very fast, for the same reasons.

1. Following our feature selection process in HW1, we chose 4 features that seemed to be highly correlated to the covid target variable.

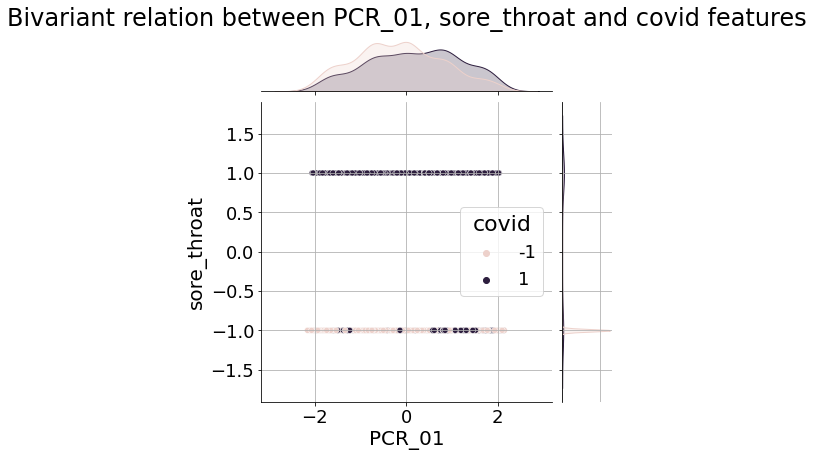
Firstly, we chose sore\_throat, as it showed high correlation to covid, as can be seen in this histogram:



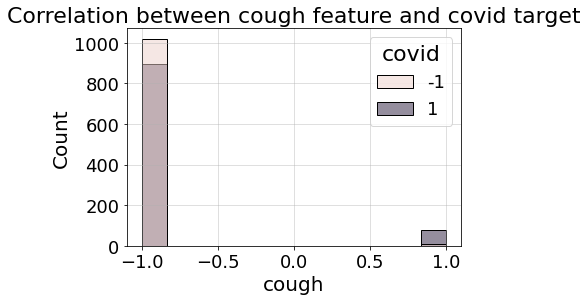
Secondly, we chose shortness\_of\_breath, as it was highly correlated to patients that had sore\_throat value of -1. This can be seen in the attached histogram:



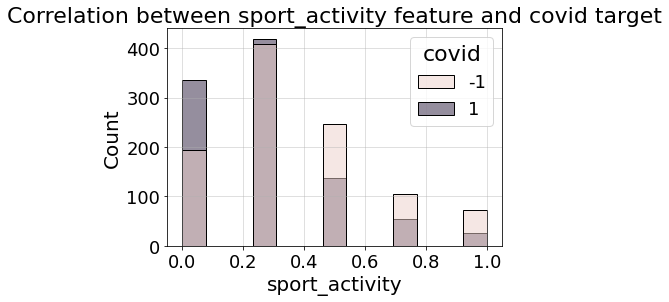
We also saw high bivariate correlation with PCR\_01 and sore\_throat, so we also chose PCR\_01 as one of the features in this question. A plot that supports this:



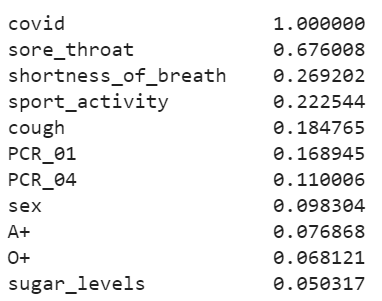
We also chose cough, as it was chosen by the sequential feature selector.



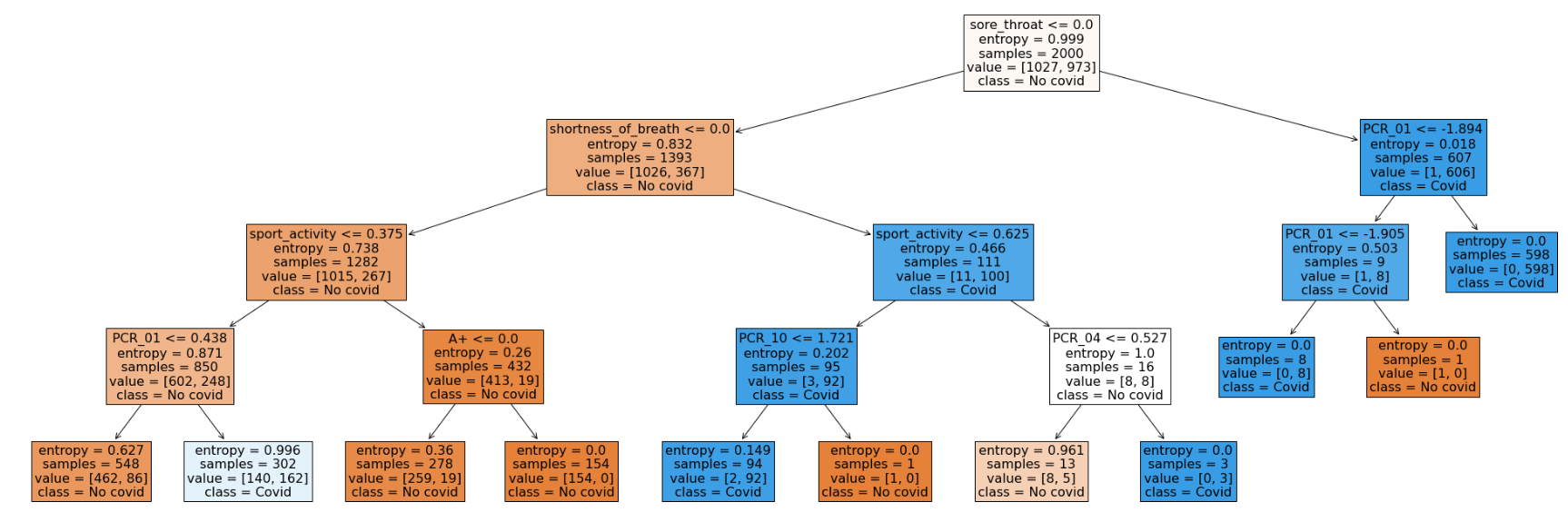
Lastly, we chose sport\_activity, since it is the third most correlated feature to covid:



1. Below are the 10 most highly correlated features to covid target variable (covid obviously has a correlation of 1 to itself, so we disregarded it):



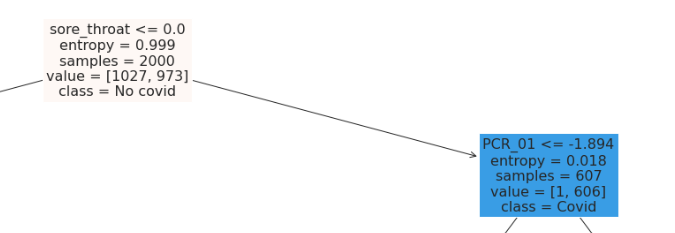
1. Below is the visualization of the trained tree:



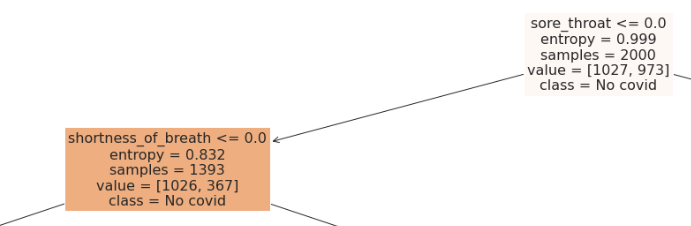
The training accuracy of the tree is 0.874.

1. We can clearly see the features we mentioned in question 7 appear at the top of the most correlated features list for covid target variable. Therefore, their importance is noticeable, and our findings from questions 7 and 8 match each other.

Regarding our tree model, we can see that the first feature used (at the root node) is sore\_throat. This matches our answer to question 7, and the fact that it appears first in question 8, showing this feature is highly correlated to covid.



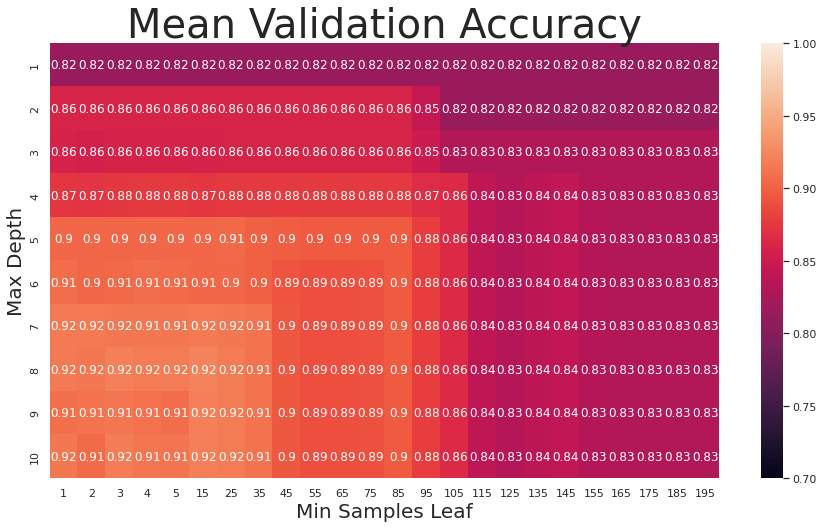
We can see that this feature separates almost a third of the overall number of samples with correct covid labeling. After that, we can see shortness\_of\_breath used on samples that had sore\_throat value of -1, exactly as we described in question 7:

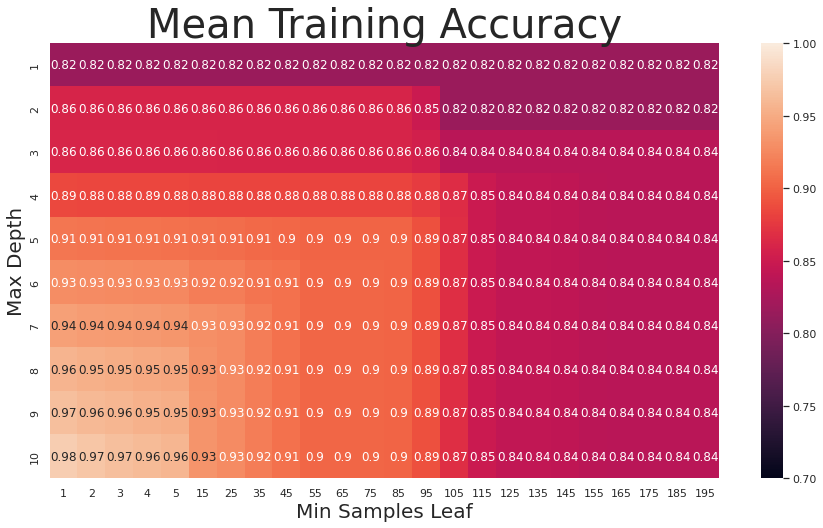


Furthermore, we see PCR\_01 is used in a further node in the tree. However, we do not see cough used in the tree model, even though both the tree and the sequential feature user use greedy algorithms. This is due to the fact that the sequential feature uses kNN as its learning model, so it uses a distance measure rather than the entropy measure used here, therefore we get different results from both algorithms.

Sport\_activity appears very early on the tree, and also fairly high on the list of most correlated features to covid, showing we were correct in choosing it as well in question 7.

1. Below are heatmaps describing the mean validation and training accuracies for different hyperparameter values:

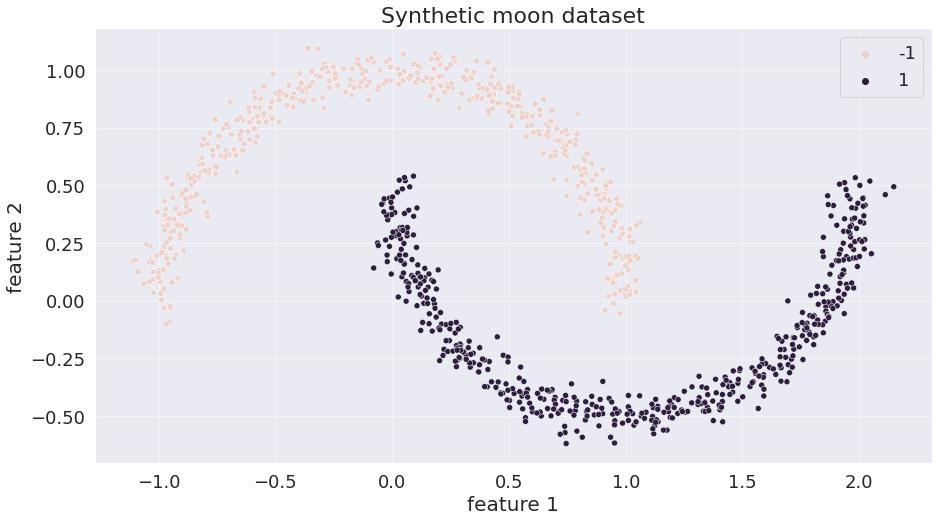




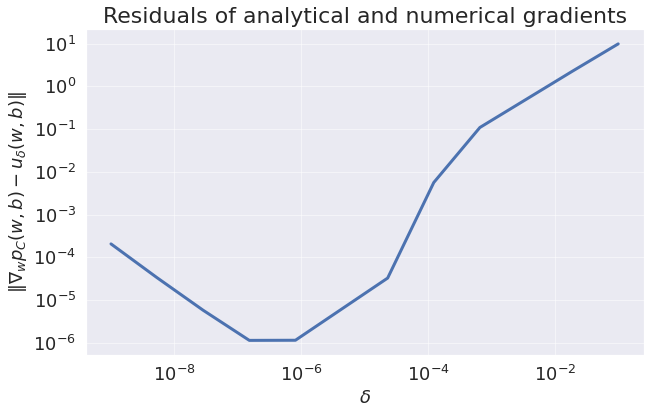
According to the values returned by the grid search function, the optimal hyperparameter values are max\_depth = 8, and min\_samples\_leaf = 15.

From the heatmaps, we can assume a max\_depth of 10 and min\_samples\_leaf of 1 would cause overfitting, and max\_depth of 1 and min\_samples\_leaf of 195 would cause underfitting. We assume the first combination of hyperparameters would cause overfitting for two reasons: Firstly, we see in the mean training accuracy heatmap that this combination yields very high training accuracy, but lower or similar validation accuracy to its neighbors, which shows the parameters are tailored for the specific training set. Secondly, these parameters result in a very complex model (a very deep tree with potentially small amounts of samples in every leaf), which we have learned is extremely prone to overfitting. Specifically in this case, a depth of 10 would result in potentially up to 1024 leaves, which for our 2000 training samples could result in 2 samples per leaf (extreme overfitting). We assume the second combination of hyperparameters would cause underfitting for exactly the opposite reasons of the first combination - we see lower training accuracy in the heatmap, and those values result in a simpler model that would struggle to tailor itself to the given training samples - a maximum depth of 1 means only 2 leaves, meaning we can only divide based on one feature.

1. The test accuracy of a tree model with the optimal hyperparameters we got is 0.908.
2. Below is the plotted moon dataset:

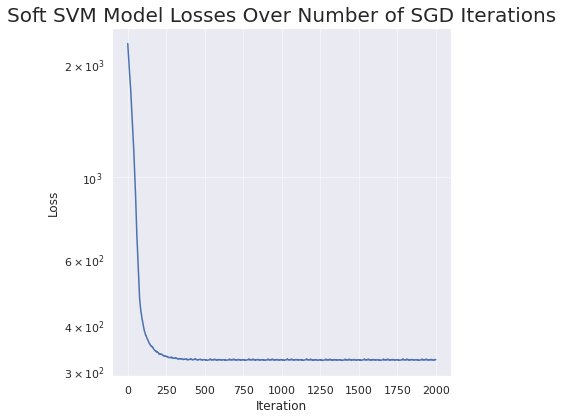


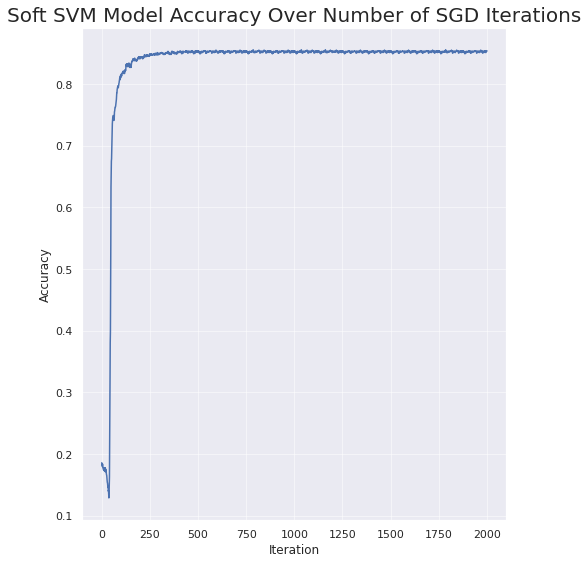
1. Below is the plot describing the comparison between the numerical sub-gradient and the analytic sub-gradient:



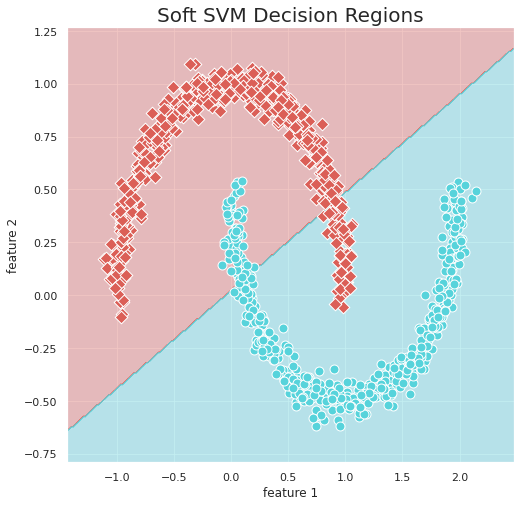
In this plot we can see that the difference between the numerical and analytical sub-gradients is fairly small for low values of delta, and that it increases as the values of delta increase. Furthermore, we can see that we don’t necessarily get the smallest difference for the lowest value of delta. This behavior is very logical - we are essentially calculating a derivative, and we know that the further away the two points we consider in this calculation (i.e. the bigger the value of delta), the more we are approximating and getting further away from the exact result. The behavior for the smallest values of delta is also logical - since the analytic and the numeric sub-gradients aren’t equal, and since for the smallest values of delta we “compare'' two vectors that are almost exactly similar, we can’t expect to get the same result for both sub-gradients. Furthermore, the smaller the values of delta is, the higher the risk we get numerical errors.

1. Below are the plots describing the accuracy and loss of the soft SVM model we created, over 2000 iterations:



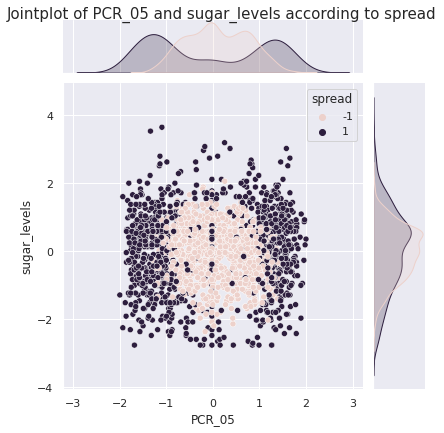


We can see that the model converges, as the accuracy improves and then plateaus - stops improving over more iterations. Similarly, the loss decreases, then stops decreasing and plateaus. We also attach the decision regions of the model:



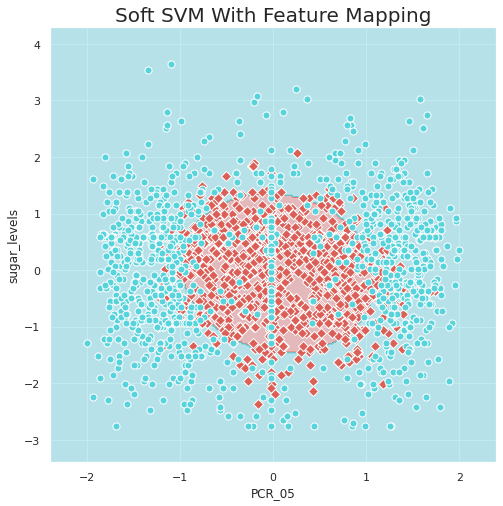
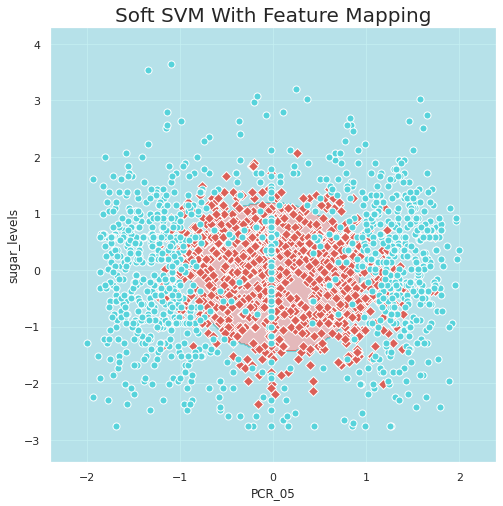
The maximal training accuracy is 0.855, achieved in iteration 593. The minimal training loss is 322.102, achieved in iteration 1265. We can see these values are attained in different steps. There are several possible reasons for this: firstly, we can see in the plots that even though we plateau, there are small fluctuations in the values of loss and accuracy starting around the 250th iteration. Because of this, we might get values that are minisculey larger or smaller than others in later iterations, even though values in different iterations at this point are virtually the same. Secondly, the goal of SGD is to minimize loss (as we see in the formula we used for implementing the soft SVM class), therefore we expect loss to decrease the more iterations we do. This could explain why the minimal loss is achieved far later than the maximal accuracy, which SGD does not actively try to minimize, but is rather minimized by the fact that the loss decreases. Finally, even though we treat loss and accuracy as complementary values, we can decrease the loss without improving accuracy - we can change the values for w and b and decrease the loss, but not change any of the labels we predict for our samples, decreasing the loss but leaving the accuracy unchanged.

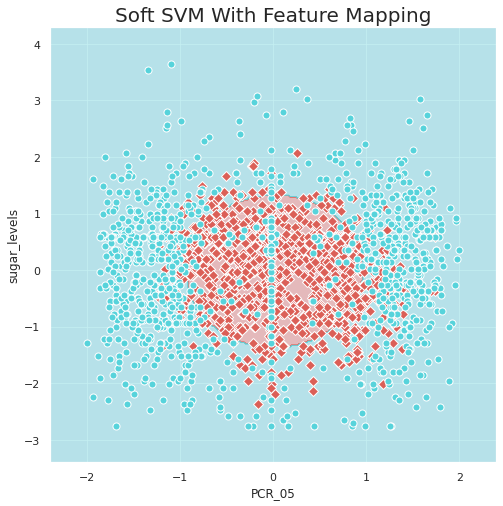
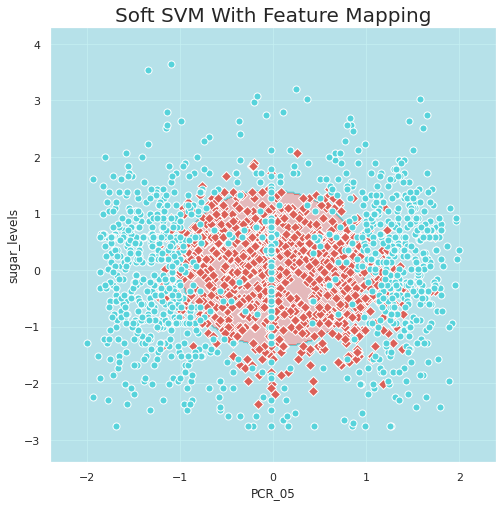
1. Below is the joinplot describing the correlation between PCR\_05, sugar\_levels and the spread target variable:

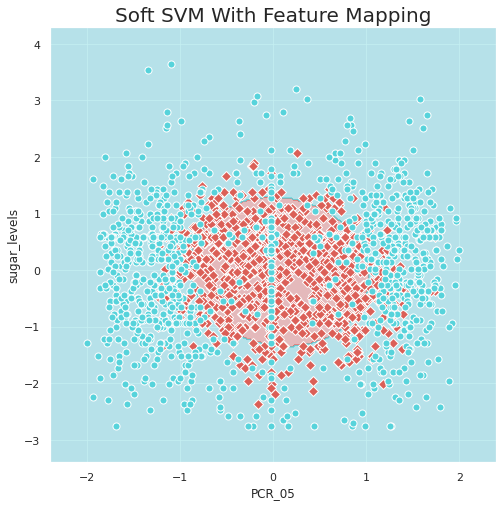


1. The training accuracies for the models are: 0.789, 0.825, 0.8145, 0.7995, 0.799. Their mean is 0.805, and their standard deviation is 0.013.

Below are the decision regions visualizations for the 5 models:

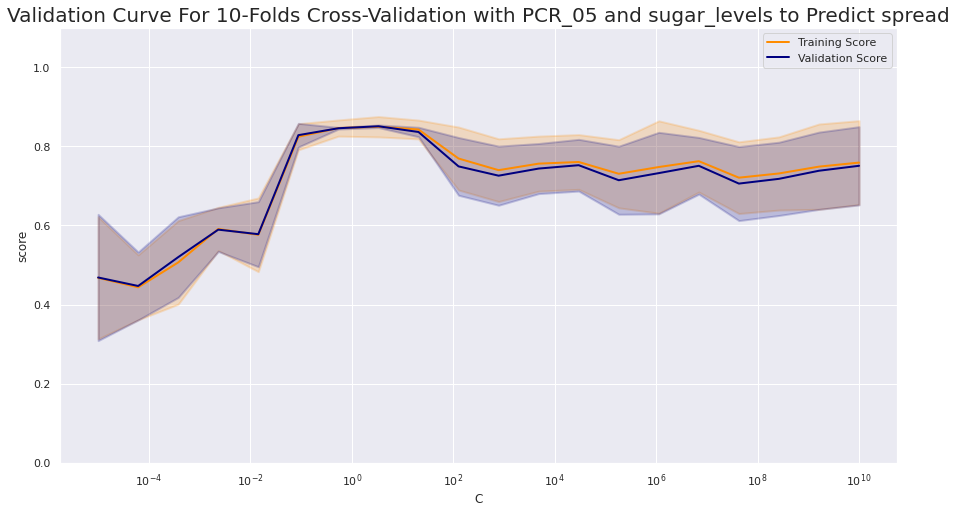






We see small variations in the results between the different models. Our SVM model uses stochastic gradient descent, and our problem is convex, meaning SGD should converge to a global minima (which is the same as local minima since our problem is convex). However, we saw in the lecture that the value we choose for lr greatly effects this convergence, with large values possibly causing us to miss the minimum point (because every step is too large), and small values possible causing convergence that is too slow (meaning we might need a larger number of iterations than the one we used). Since we didn’t fine tune our lr value, either one of these problems might occur here. Furthermore, since SGD is initialized with a random vector, and we use the same lr value for all models, it is logical we converge to a different value every time (since we just explained why we don’t converge to the minimum point every time).

1. The range we chose for C value is . Below is the validation curve for the 10-folds cross-validation we performed:

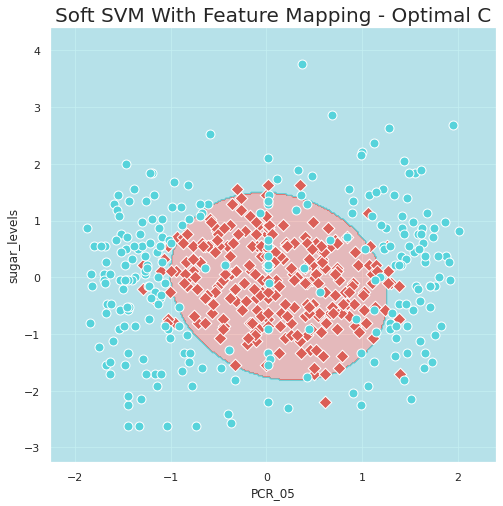


In the validation curve we see different behaviors for different values of C: for lower values of C, we see low validation accuracy (under 0.6). The reason for this comes from the formula for soft SVM - if we chose a low value for C, the model puts emphasis on making the norm of w smaller, rather than minimizing loss. Because of this, we get higher loss and therefore lower accuracy. For higher values of C, we see higher validation accuracy, but not as high as the optimal results. The reason for this is that these values of C cause the model to put emphasis on minimizing the loss, but no emphasis on minimizing the norm, therefore we get w with a higher norm which can cause overfitting. For the middle values of C we get optimal validation accuracy - we put emphasis on minimizing loss, but not so much as to allow the norm of w to be too large.

We also see in this validation curve, that the values for validation and training accuracy are fairly similar. This can come from the fact that we used 10-folds and not 8-folds, meaning we trained on a larger part of the data every time.

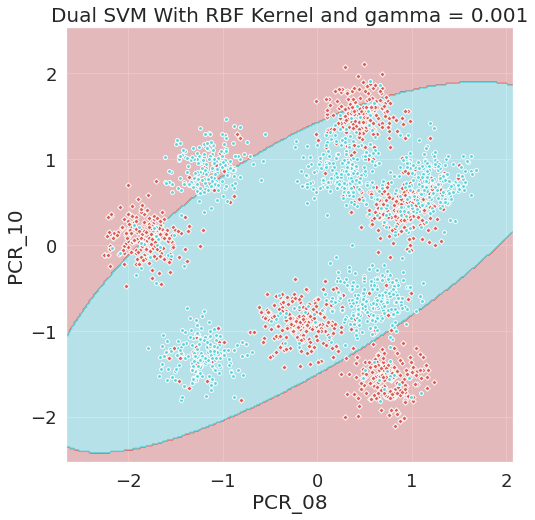
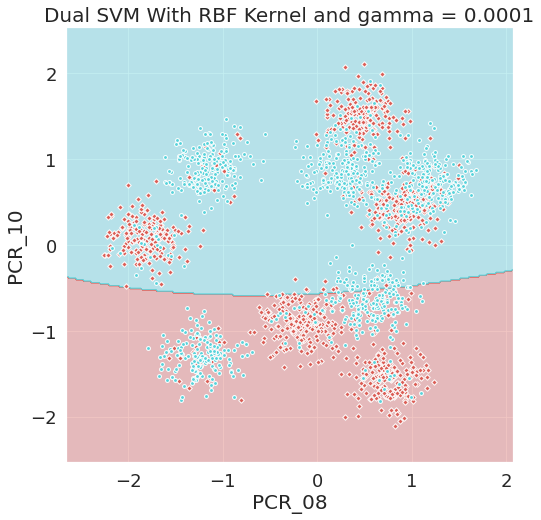
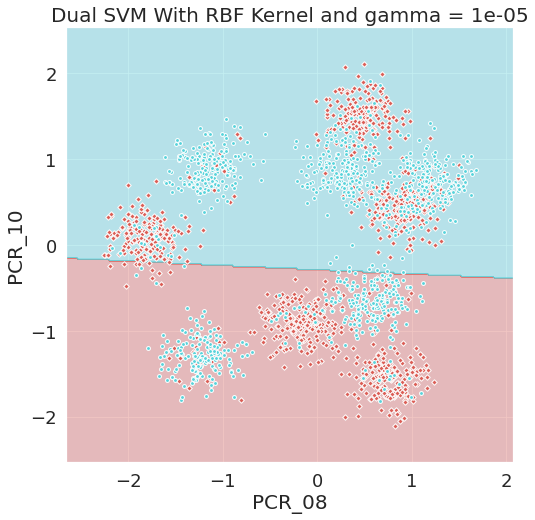
The optimal C value is 3.36, for which we get a mean training accuracy of 0.85 and a mean validation accuracy of 0.851.

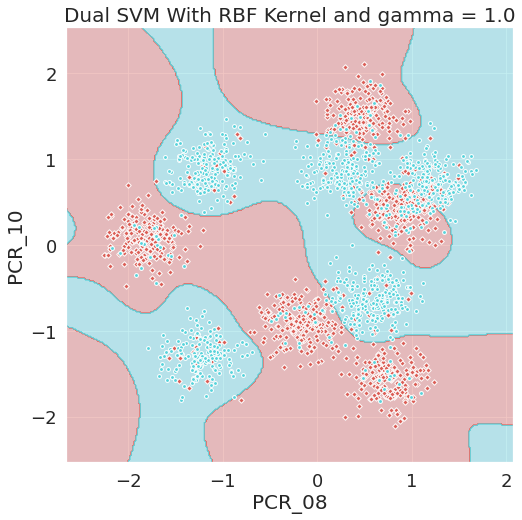
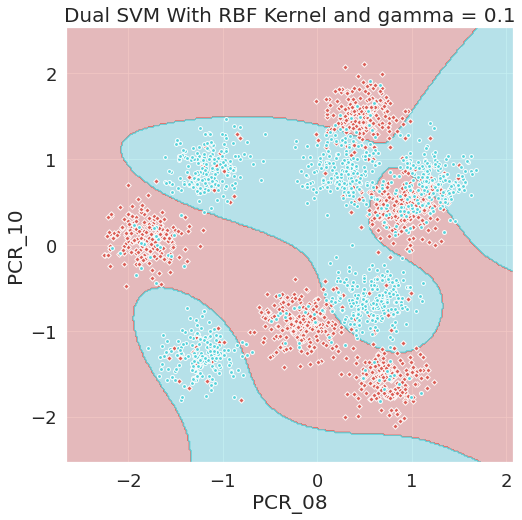
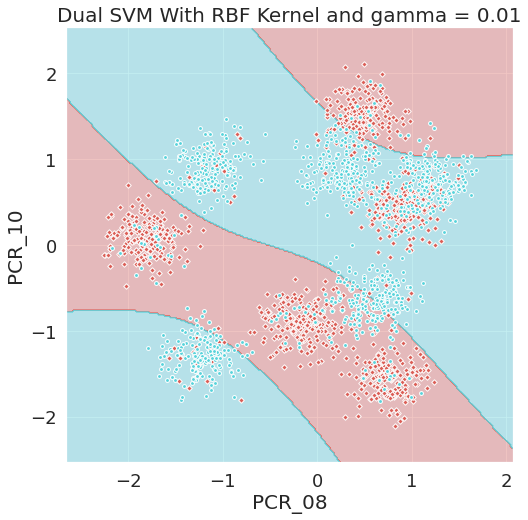
1. Below are the decision regions for soft SVM with the optimal C value:

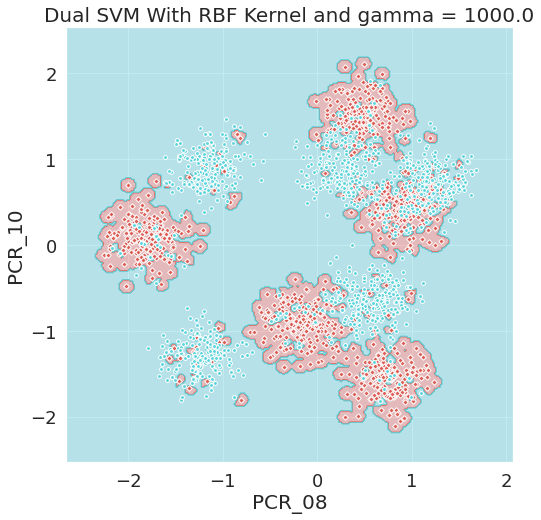
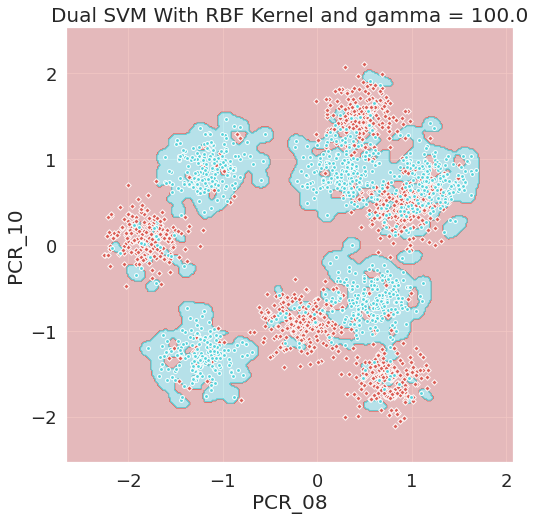
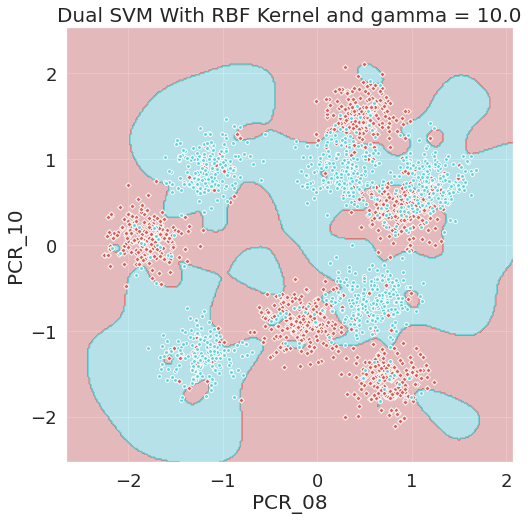


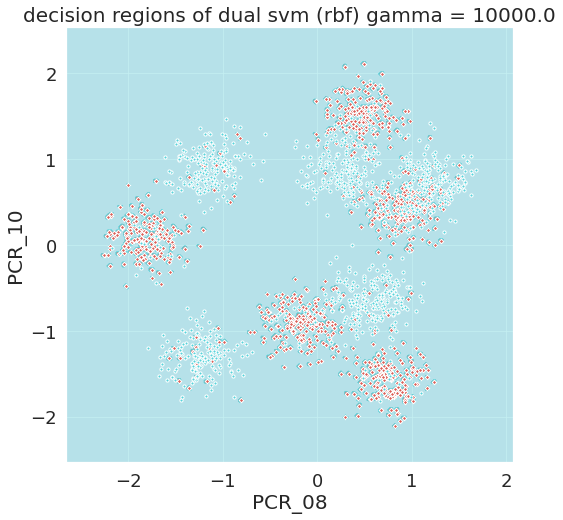
The model’s test accuracy is 0.878.

1. Below is the visualization of the decision regions of the ten SVM models with RBF kernels, with different values of gamma:

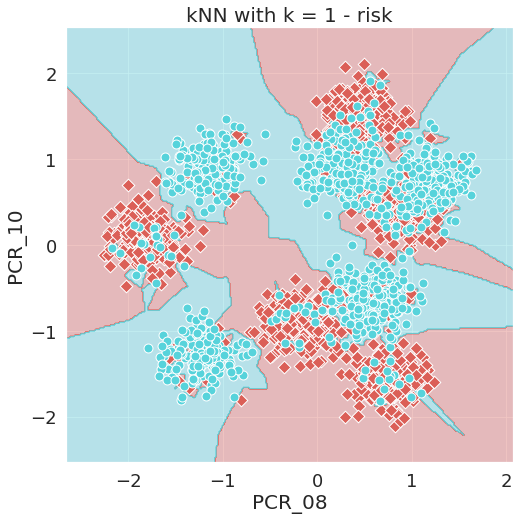
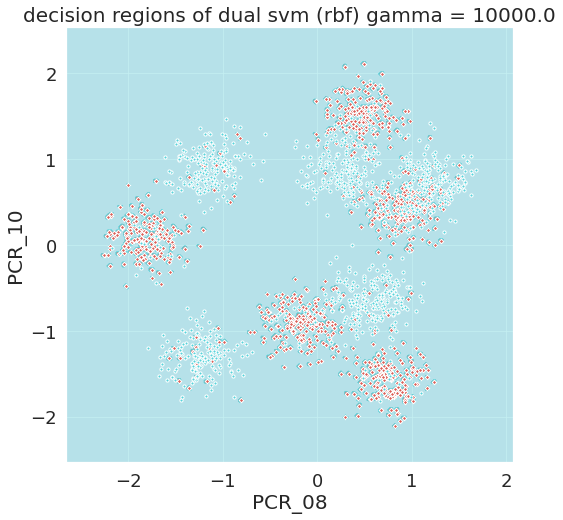






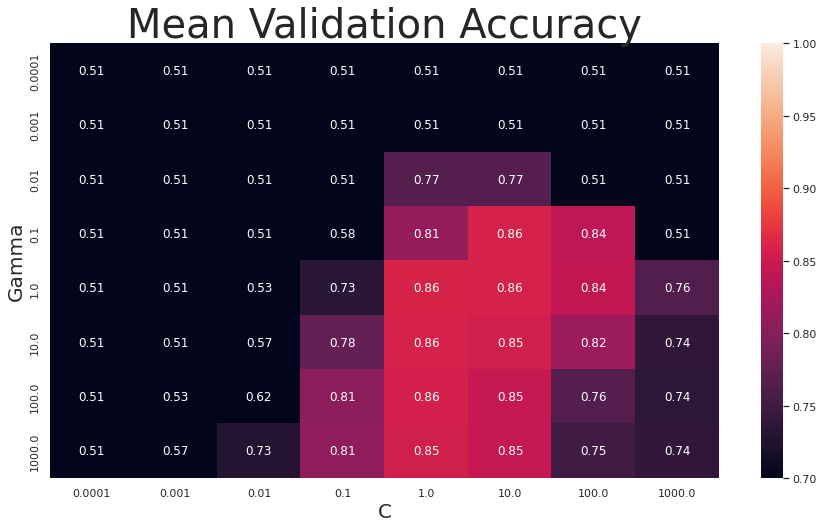


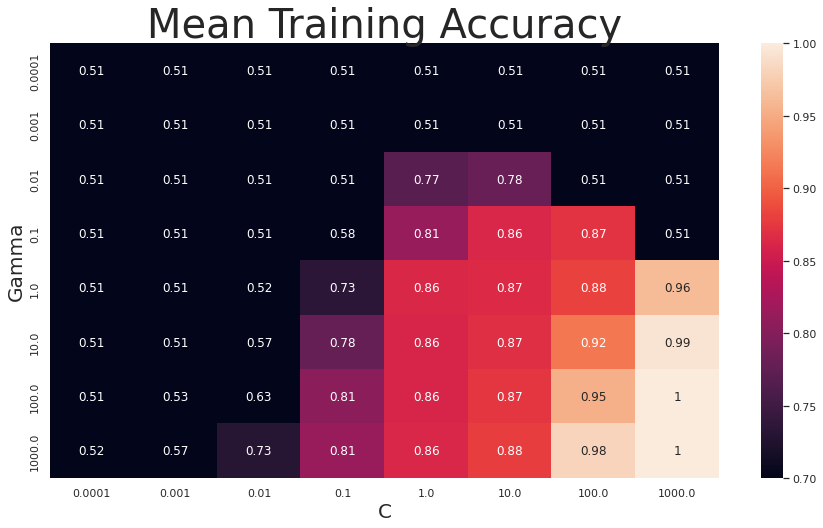
1. Below we compare the plots for soft SVM with RBF kernel and gamma value of 10000, with knn with k = 1:



Since we used a very high value for gamma, we expected the model to act similarly to knn with k = 1, since the gamma value causes the model to approximately only consider the closest sample to the sample being checked - the same as knn with k = 1. However, we can see the models return different decision regions. This could be due to the fact that knn with k = 1 only takes into account the closest sample, while our rbf kernel model approximates this behavior, but still somewhat takes into account samples that are not the closest one. Since we have big groups of points clustered together, each sample would still be affected by several other samples. Another possible reason is that knn uses the distance measure as it is, and the rbf kernel uses it exponentially, causing different results.

1. Below are heatmaps describing the mean validation and training accuracies for different hyperparameter values:





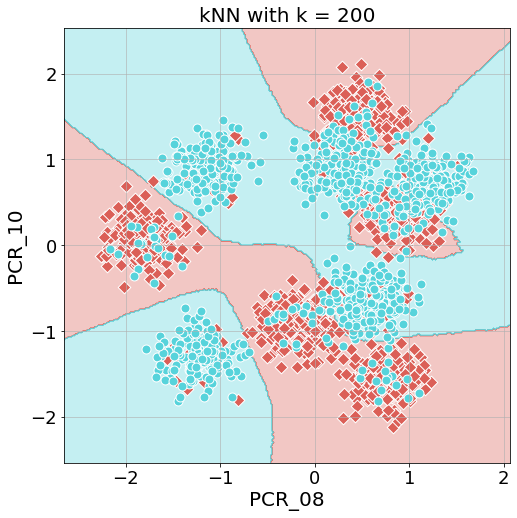
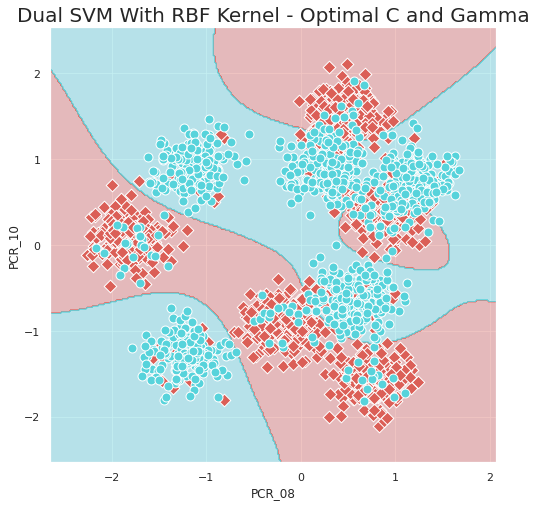
According to the values returned by the grid search function, the optimal hyperparameter values are C = 1, and gamma = 1.

From the heatmaps, we can assume C = 1000 and gamma = 1000 would cause overfitting, and C = 0.0001 and gamma = 0.0001 would cause underfitting.

The values we mention would cause overfitting for two reasons: Firstly, a large C value would cause us to disregard the norm of w almost completely, and to focus only on minimizing loss for the specific given training data, causing overfitting. Secondly, a large gamma value would cause the model to behave similarly to knn with k = 1 (as explained in the assignment), leading to overfitting - the prediction for each sample would approximately only take into account the closest sample from the training data (overfitting).

The values we mention would cause underfitting for two reasons: Firstly, a small C value would cause the model to focus mainly on minimizing the norm for w, and would allow him larger losses, therefore partly disregarding the training data - underfitting. Secondly, a small gamma value would cause the model to take into account a larger amount of other samples every time, causing underfitting (same as knn, we take into account samples that are not even similar to the predicted sample).

1. Below are the decision regions for the optimal values for C and gamma, compared to the decision regions we got in question 4 (for knn with k = 200):



For the SVM model with RBF kernel, we got a test accuracy of 0.842. This is very similar to the test accuracy of 0.85 we got with knn. However, we can see the boundaries are smoother for the SVM model, which would make the SVM model slightly preferable. Since their behavior seems to be fairly similar, it’s difficult to say which one would work better at predicting labels for our data, but we can say that if we’re considering more then a few features - RBK kernel SVM would be preferable and would work better, and if we’re considering a small number of features - knn would be preferable.