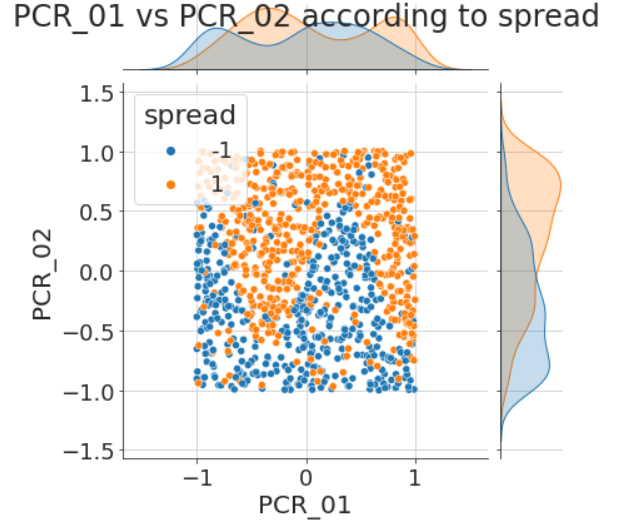
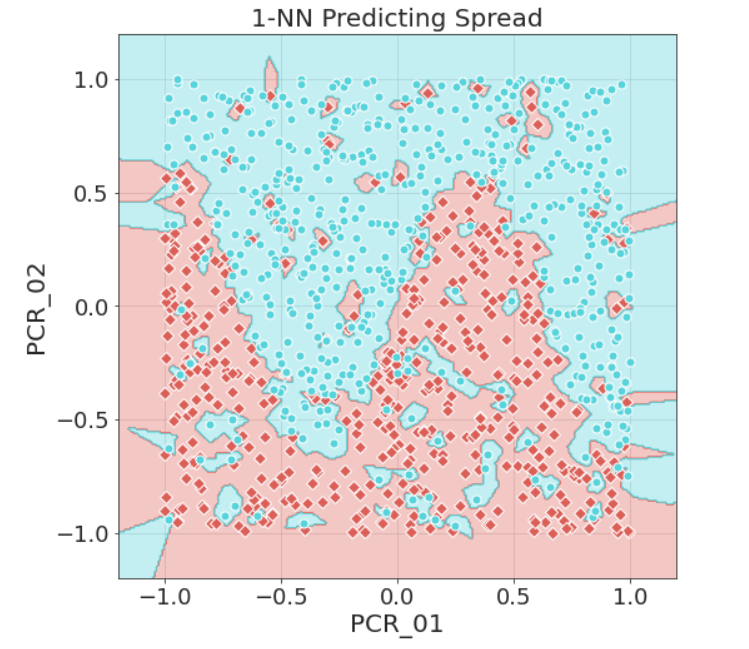
# **Major HW2 – Final Report**

## (Q1)

The following is a jointplot of PCR\_01 and PCR\_02 according to spread:

## (Q2)

The decision regions of a 1-NN classifier:



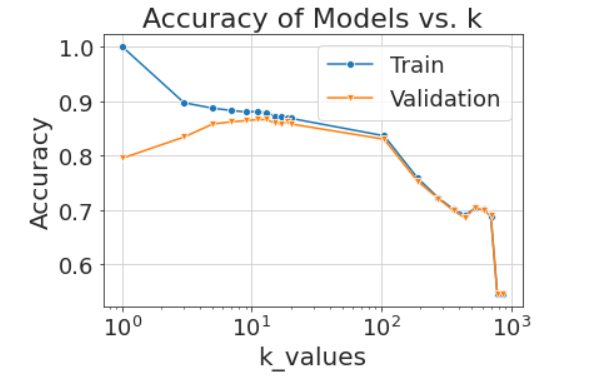
spread:

1

-1

## (Q3)

The following is a plot of the accuracy of the models vs. the value of k (where k is on a logarithmic axis):



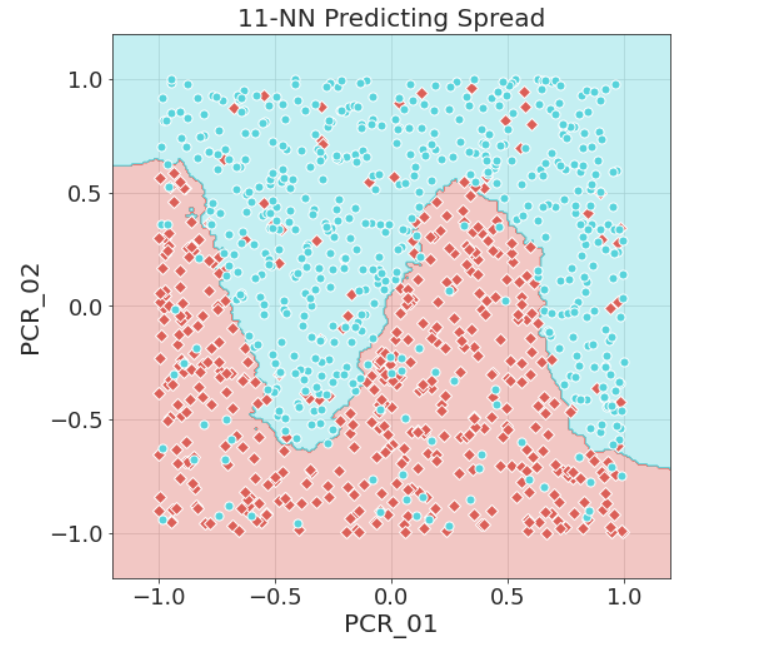
The best k value is 11 with a mean validation accuracy of 0.866 and a mean train accuracy of 0.881.

Low k in kNN values cause overfitting – since few neighbors are taken into account, the influence of each neighbor is high, and therefore outliers in the training set can create a region of wrong classification around them. This can be seen in the plot – the train accuracy is high but the generalization is not as good (validation accuracy is lower).

High k values cause underfitting – as k gets large, far samples are taken into account equally as neighbors, even though their relevance is low, and so the information embodied in the training set is not utilized. This results in low validation accuracy. It is interesting to note that training and validation accuracies are virtually the same for large k’s – this is reasonable since an underfit model barely refers to the specific training set it is trained on (so the behavior on the validation set is similar).

## (Q4)

The following plot shows the decision boundaries for the optimal k=11 we found:



spread:

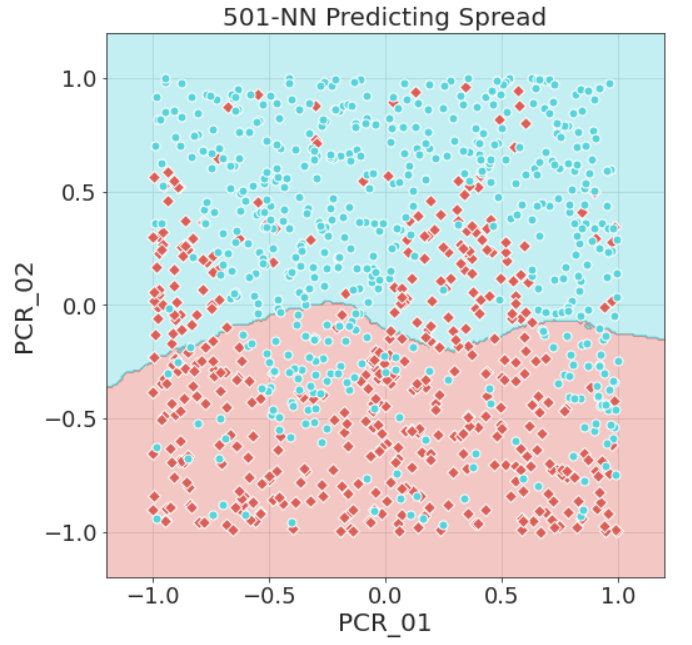
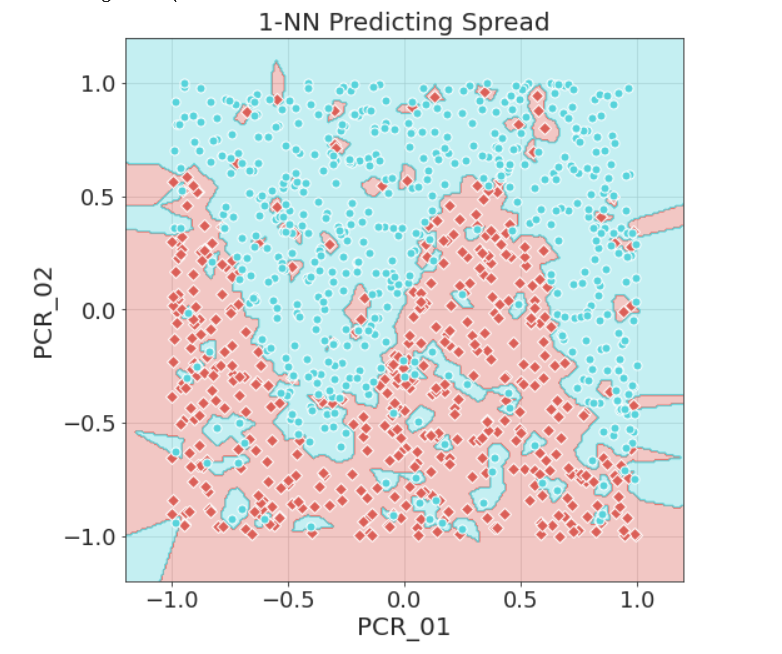
1

-1

The accuracy of the model on the test set is 0.892.

## (Q5)

We trained two additional models with k=1, 501:



spread:

1

-1

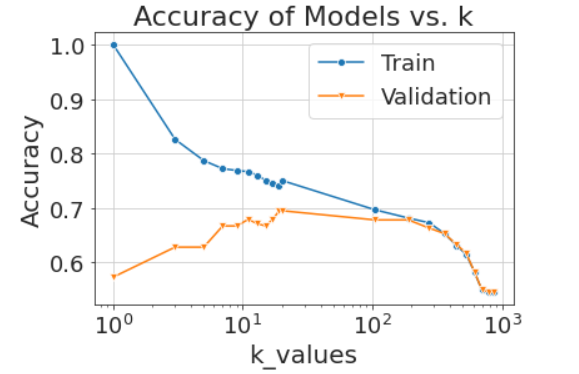
The decision boundaries of these models behave as we explained:

For k=1, the lowest possible value for k, every point creates a decision region around it. Therefore, all outliers create decision regions of wrong classification.

However, k=501 is a very large value for this dataset: for comparison, the whole training set has 1000 samples, so each point takes into account as neighbors roughly half of the samples! For example, the large concentration of spread=-1 (red) values around is wrongly classified as 1 (blue) since it is surrounded by mostly blue points, in the upper half of the plane, and since k is large, we take into account more of the surrounding points than the points of the concentration itself.

## (Q6)

The following plot shows the mean training and validation scores of k-NN models using the whole feature set:

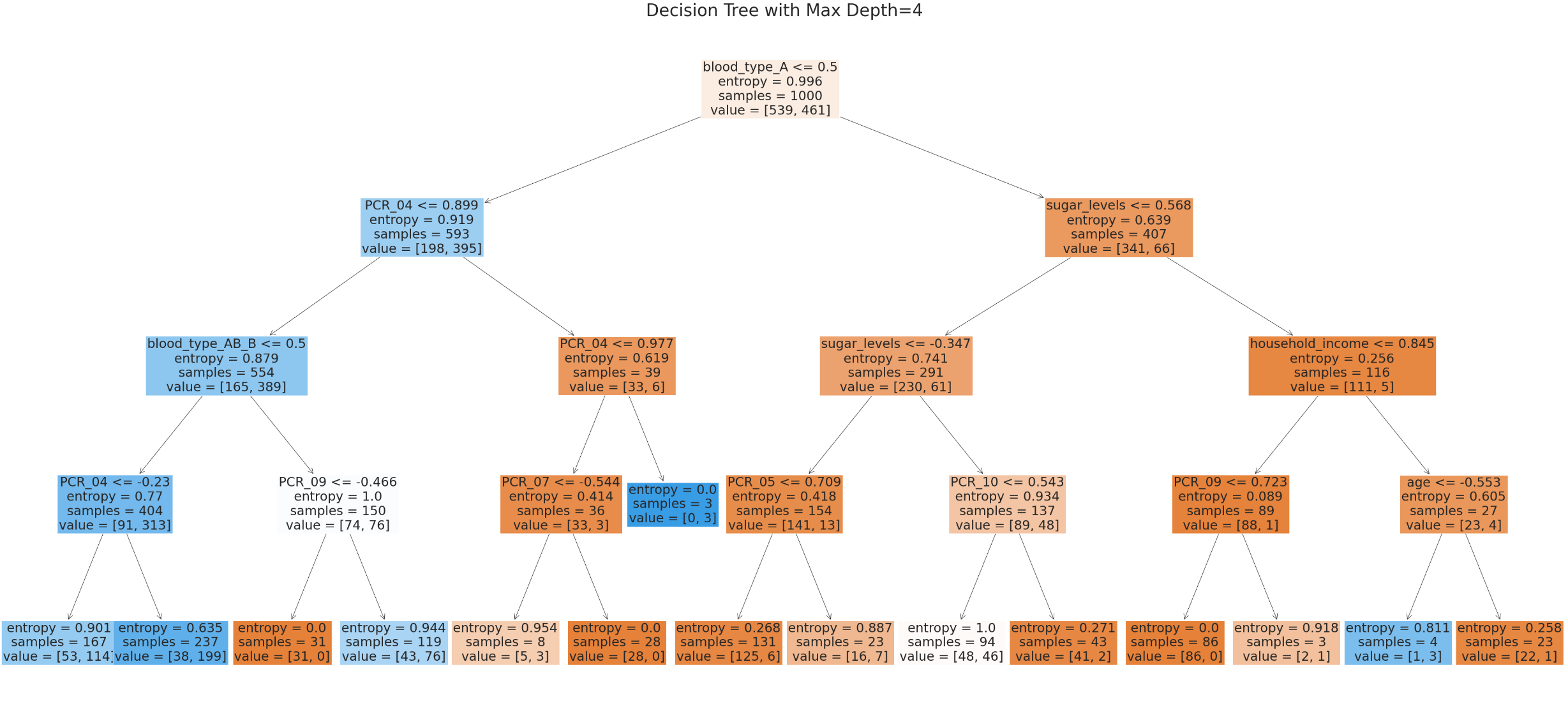


As we can see, these models are significantly worse than the PCR\_01 x PCR\_02 models: the best validation accuracy in this case is approximately 0.7, compared to 0.866 previously.

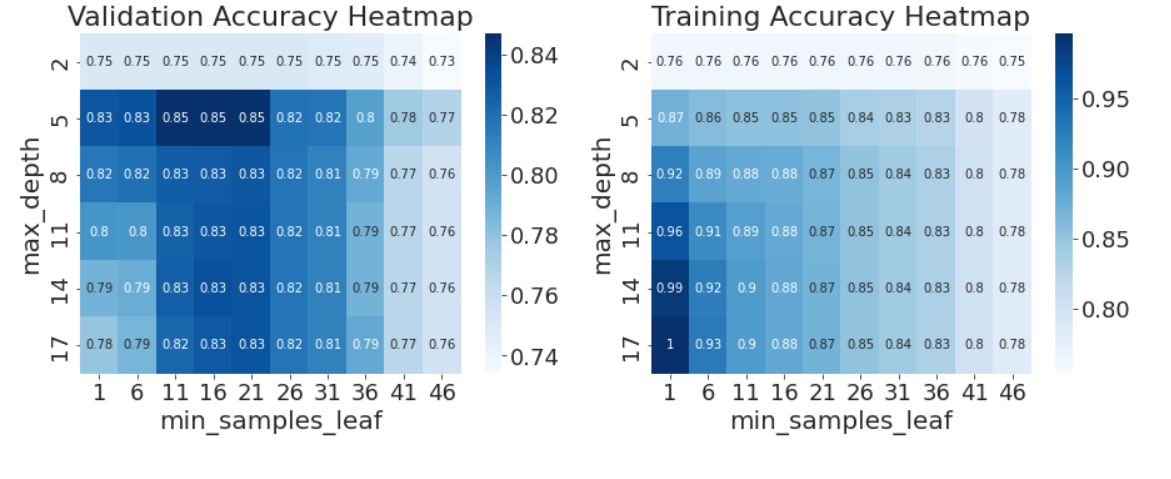
The reason behind this, is probably that most features in the dataset are not appropriate for k-NN models – proximity in regard to the feature does not predict equality in regard to the *spread* label (for example – the *sex* feature). Since these features affect the distance calculated between each two samples, but their distance does not correlate with the target label, the altogether prediction is harmed.

However, it should be noted that the overall trend of the train and validation plots is similar to (Q3) – since low k values still lead to overfitting, and for high k values we receive underfitting and merging of train & validation accuracies.

## (Q7)

The training accuracy we received for the ID3 Decision Tree model with max\_depth=4 was 0.799.

## (Q8)

After searching for appropriate parameter ranges, we choose *min\_samples\_leaf* and *max\_depth* . We obtained the following heatmaps:

The optimal combination for validation accuracy is *max\_depth*=5 and *min\_samples\_leaf* of 11-21.

An example hyperparameter combination that caused **underfitting** is *max\_depth*=2 and *min\_samples\_leaf*=46.

An example hyperparameter combination that caused **overfitting** is *max\_depth*=17 and *min\_samples\_leaf*=1.

The reasons for this behavior:

Low values of *max\_depth* lead to underfitting since few decision are made regarding the data – there probably remain more useful splits to execute on the data. Similarly, high values of *min\_samples\_leaf* also cause underfitting since they stop the splitting process, even if the entropy of the node is high.

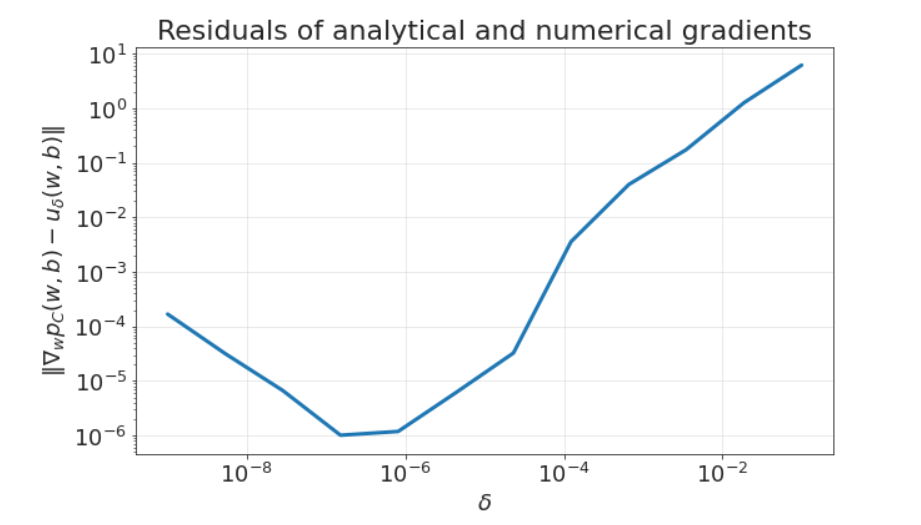
On the other hand, high values of *max\_depth* and low values of *min\_samples\_leaf* lead to overfitting since they allow extensive splitting in a way that fits small amounts of samples, even if they are outliers, therefore highly fitting the training set and causing low generalization.

## (Q9)

We trained a decision tree with *max\_depth*=5 and *min\_samples\_leaf*=16 and obtained a test accuracy of 0.828.

## (Q10)

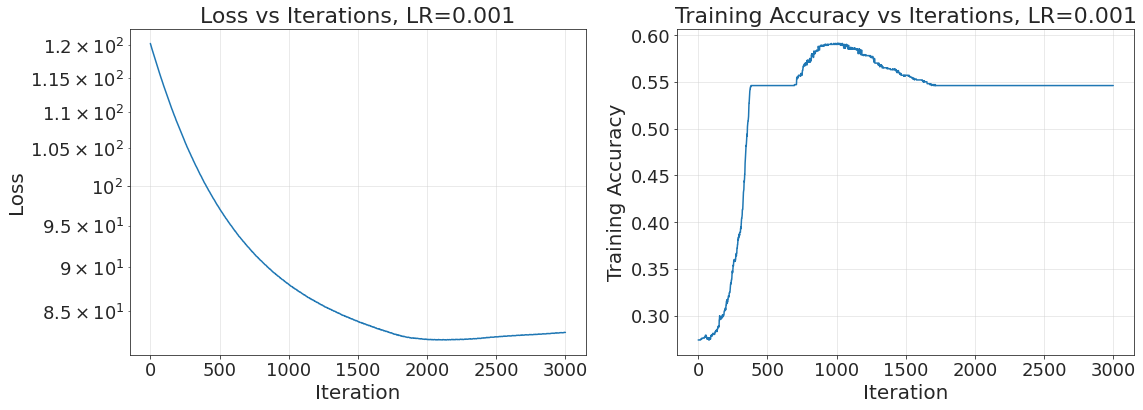
The following plot describes the residuals of the numerical and analytic subgradient:

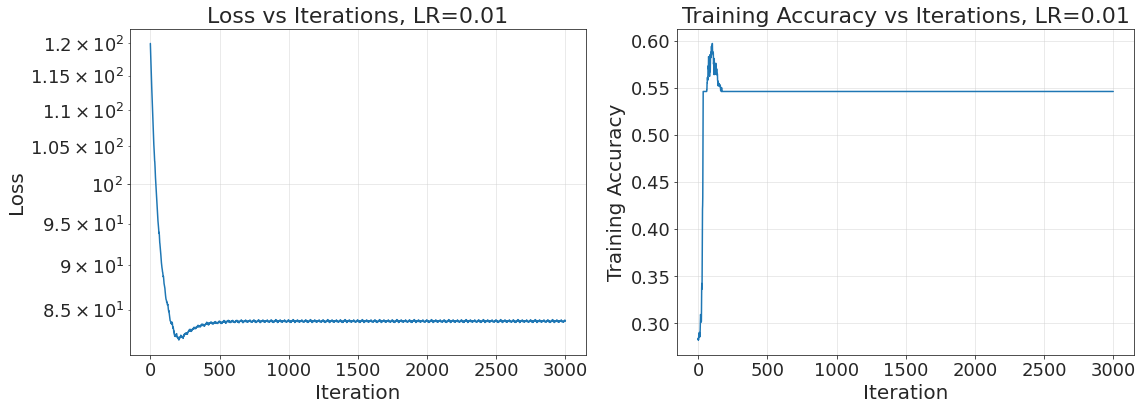


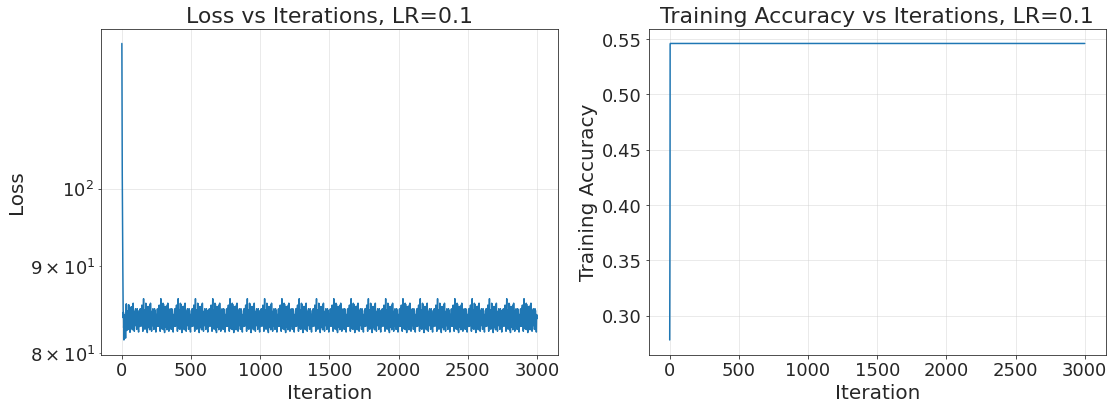
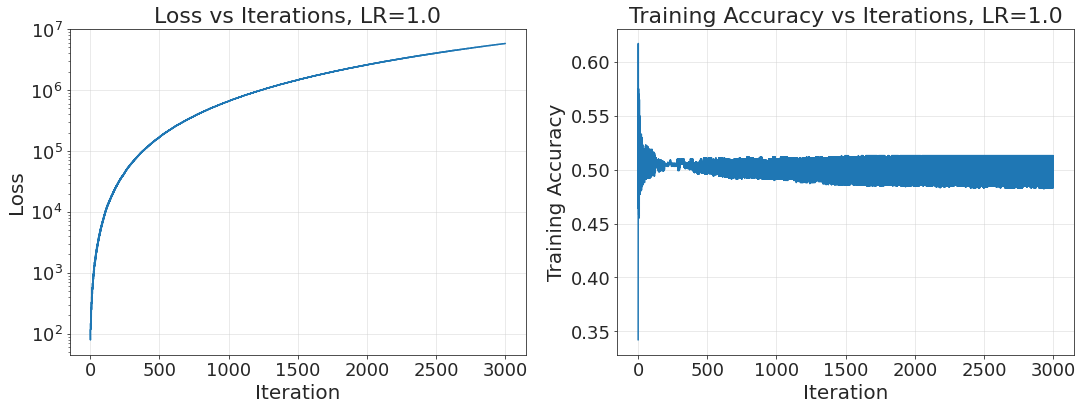
As we can see, as gets smaller, the residuals also converge to zero – this is because the numerical subgradient converges to the analytic formula as goes to zero. Note that for very small values of () the residuals grow; most likely this is due to numerical errors in calculating the numerical subgradient.

## (Q11)

We trained an SVM model with learning rates 0.001, 0.01, 0.1, 1, and received the following loss and accuracy plots:



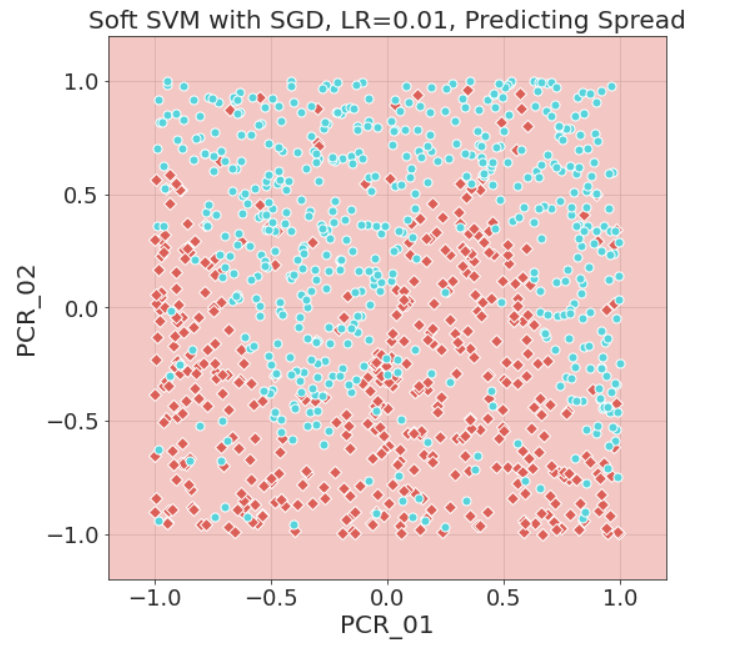




a. Based on the above plots, the learning rate we would choose is 0.01:

* First, we can notice that all three rates 0.001, 0.01 and 0.1 converge to approximately the same loss value () and accuracy . In contrast, for learning rate 1 the loss completely diverges – probably the step size is too large, which makes get farther away from the global minimum with every iteration.
* With LR=0.001 the convergence is very slow – we reach the loss minimum around 2000 iterations, compared to iterations for LR=0.01 and very few (appears ) iterations for LR=0.1. Therefore we prefer one of 0.01, 0.1.
* The behavior of LR=0.01, 0.1 is similar, but for LR=0.1 we have larger oscillations around the minimum. It is still enticing to choose LR=0.1 because of the very fast convergence, but we preferred to allow ourselves a few hundred iterations to reach a more precise minimum.

b. The following plot shows the decision regions for LR=0.01:



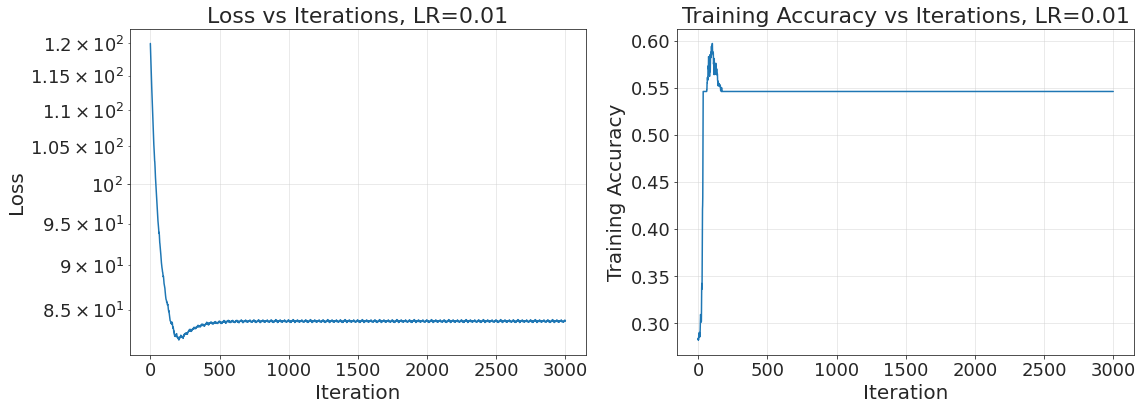
spread:

1

-1

We can see that the resulting classification is a majority rule (this makes sense since our final accuracy was around 0.55). Clearly we would like to do better in the next questions 😊

c. Let us look again at the loss and accuracy plots:



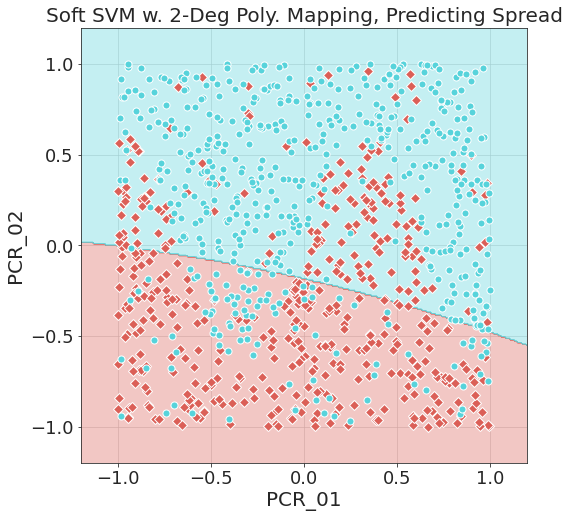
By checking the loss and score logs from the algorithm, we obtained that the minimum loss was 82.07 in iteration 190 and the maximum accuracy was 0.59 in iteration 104. They are not attained at the same step: this is because the loss function includes both the hinge loss calculations (which are inversely related to the accuracy) **and the regularization.** Therefore, which are a minimum point for the loss function are not a maximum point for the accuracy.

(Note: the hinge loss, that appears in the SVM objective function, is not identical to 0/1 loss, which is used when calculating accuracy, **but** minimizing hinge loss minimizes 0/1 loss as well since hinge loss is an upper bound on 0/1 loss)

Create softsvm.py

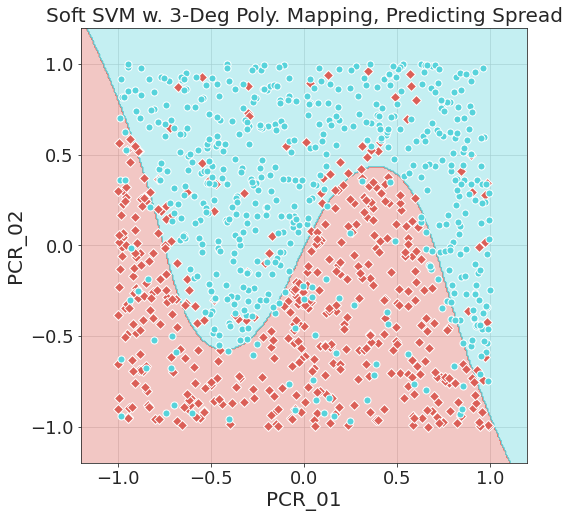
## (Q12)

We trained a Soft-SVM model on the data after two feature mappings: 2-deg polynomials and 3-deg polynomials.



Training accuracy: 0.73

Test accuracy: 0.756



Training accuracy: 0.845

Test accuracy: 0.84

spread:

1

-1

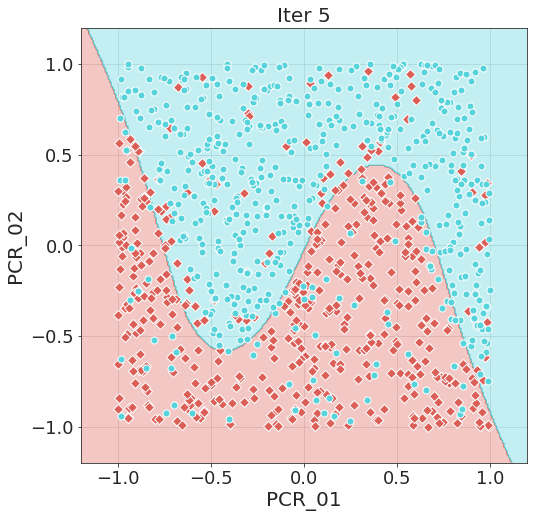
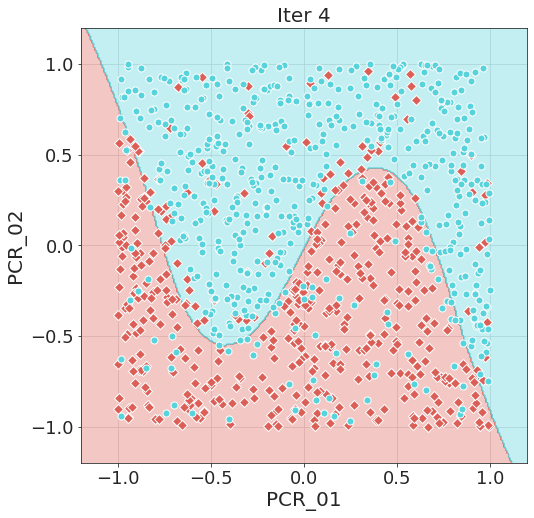
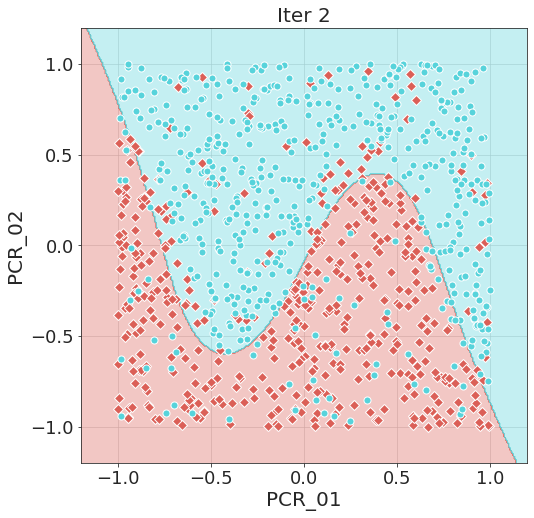
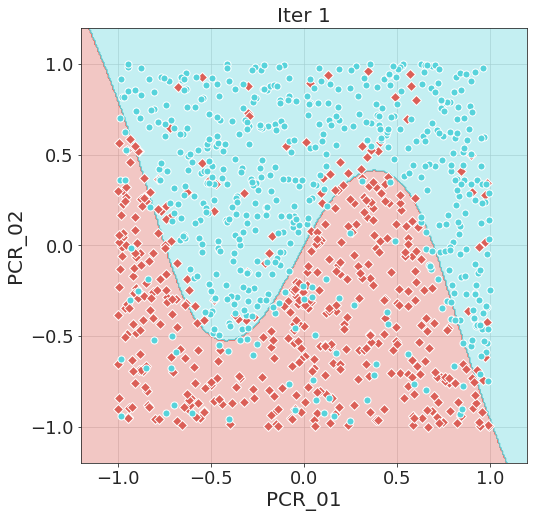
We can clearly see that the 3rd degree polynomial feature mapping has much better accuracy results and our visual impression is that it fits the distribution of the data well.

We can explain this based on the theory of feature mappings: the optimization helps us find the optimal coefficients of a linear combination of the components of the feature mapping – in our case, we map to monomials up to degree , which means in effect that we are trying to fit a -degree polynomial in variables such that its sign predicts our two classes.

A 3rd-degree polynomial is much more expressive than a 2nd-degree polynomial, and specifically for our data, we have a “wavy” curve with two extremums which is unsuitable for a 2nd-degree polynomial but suitable for a 3rd-degree polynomial.

## (Q13)

We ran five iterations of Soft-SVM with the 3rd degree polynomial mapping:



spread:

1

-1

Soft SVM w. 3rd-deg Poly. Mapping, Predicting Spread

Five iterations

The training accuracies we obtained (by order): 0.841, 0.844, 0.838, 0.836, 0.843.

Mean: 0.8404

Standard deviation: 0.003

As can be seen from the visualization and from the standard deviation of the accuracies, the variability of the models is very low. We can explain this since we know that after the feature mapping, we are still solving a Soft-SVM problem, and we know that the Soft-SVM objective function is convex. This means it has a single global minimum, so, **if we choose a correct learning rate** we are assured to converge close to this minimum with Gradient Descent.

Still, there is **some** variability, which comes from:

1. Different starting points (in our implementation the starting point is chosen randomly)

2. Random batches – the SGD algorithm calculates the sub-gradient at each step using only a batch of fixed size. In our implementation we randomize a permutation on the data in the beginning of the algorithm, so in different runs we have different orders for the samples and therefore different batches which leads to different steps.

## (Q14)

Let us observe the behavior of RBF when . Let , be our sample set, dual coefficients, and labels, respectively. Let be some point we wish to classify, and denote . So:

Since ,

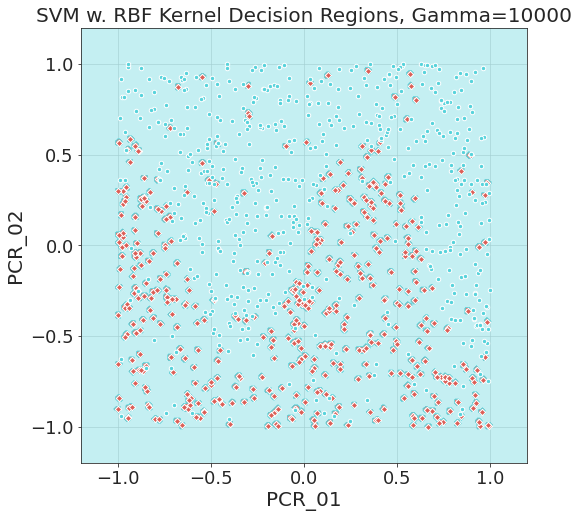
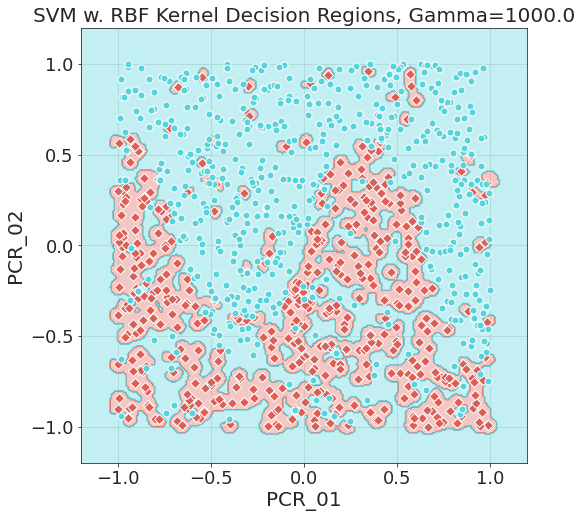
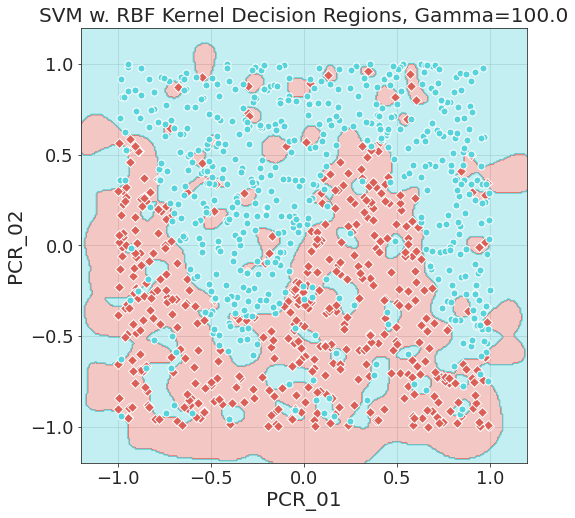
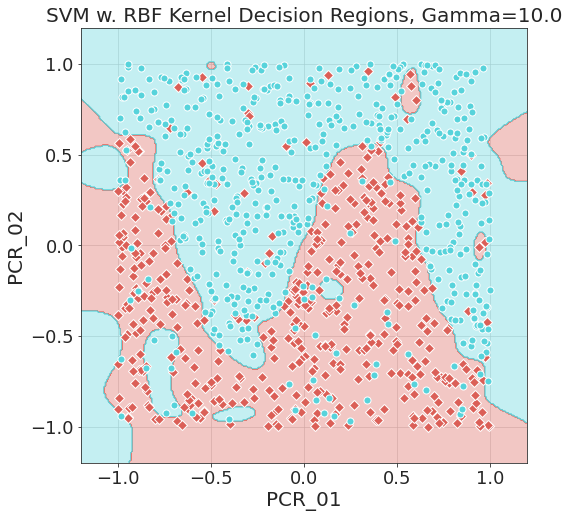
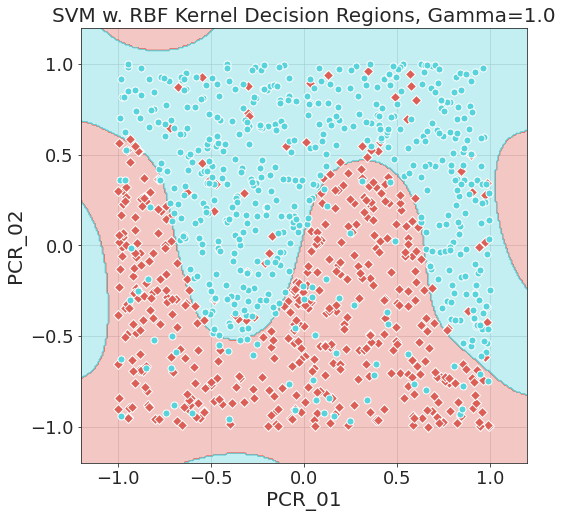
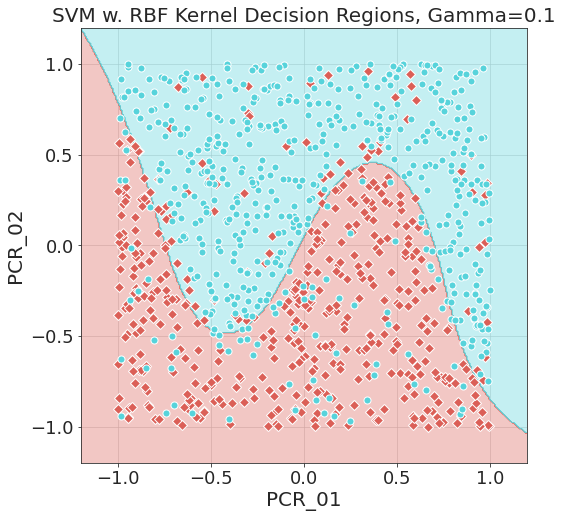
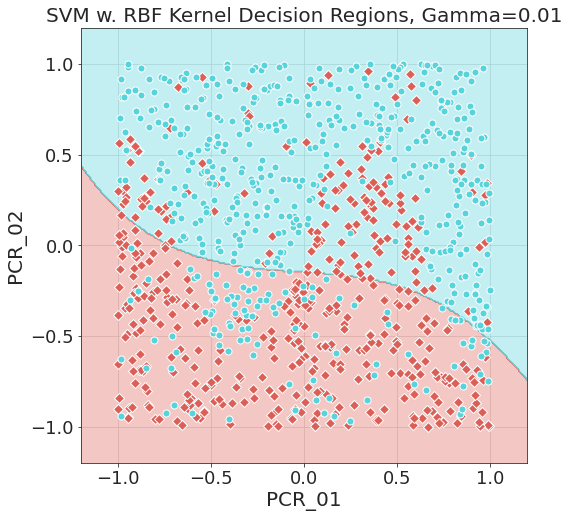
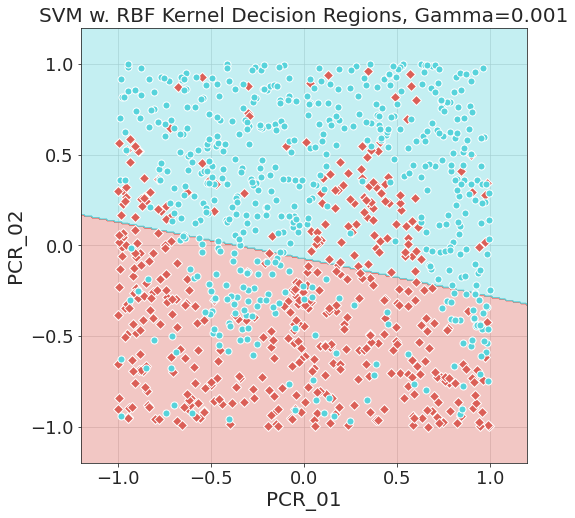
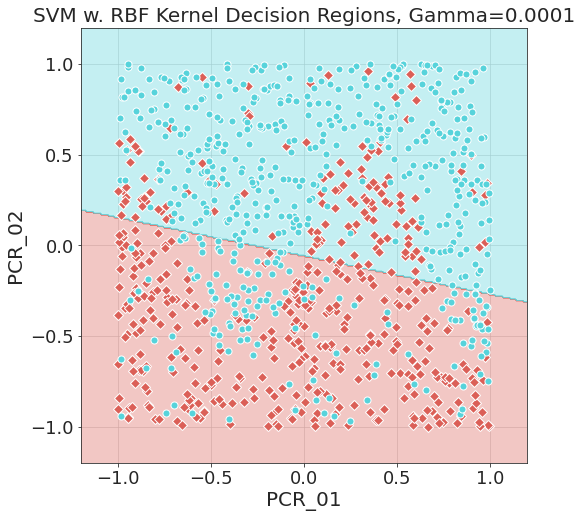
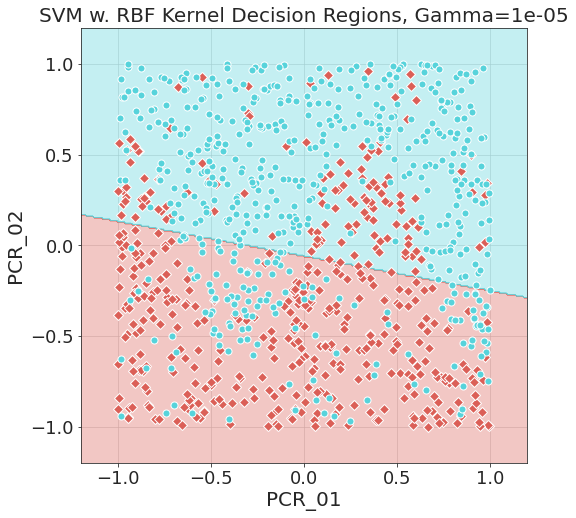
As , since for every (according to the simplification assumption that there are no two points of equal distance), , and therefore .

So

As required.

## (Q15)

The following plots show decision regions for SVM with a RBF Kernel for different gamma values:



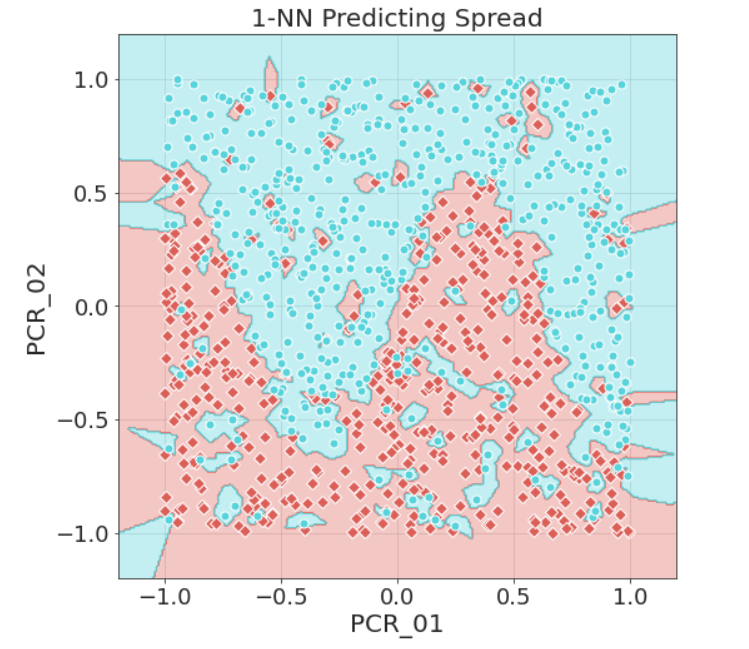
spread:

1

-1

## (Q16)

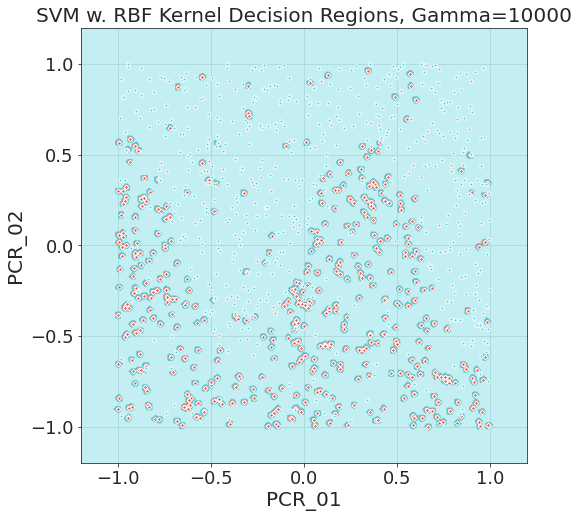
Let us see them side by side:



spread:

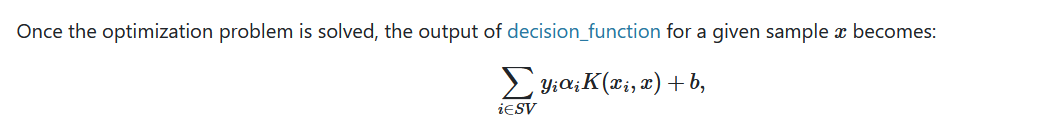
1

-1



We expected from the theoretical derivation to obtain a model similar to 1-NN for very large gamma values in the SVM RBF kernel. However, we got a model in which the “red” (-1) decision boundaries are very small and concentrated around the read points, while the “blue” (1) decision regions dominate most of the plane.

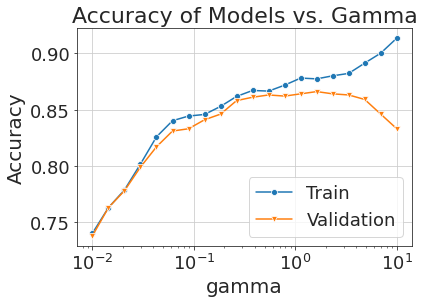
We think the reason for this is **the intercept b** in the formulation of the SVM problem in the library we are using. According to the [documentation](https://scikit-learn.org/stable/modules/svm.html#svm-kernels),



The optimization of the dual SVM problem chooses the dual coefficients and the intercept to bring the dual objective function to a minimum. Notice that *b* was not present in **our** formulation of the problem.

Now, what occurs is that *b* obtains some value – in the case of our plot above, *b=0.08*. When the decision function is calculated, is extremely small almost on every point of the plane and for any training sample . Therefore, and since also obtains some fixed value which does not come close to compensating for , the dominating factor in the decision factor becomes , unless one is very close to a training sample (and as grows, one needs to get exponentially closer). Since in our case is positive, most of the plane is colored blue, and the model does not exhibit a 1-NN behavior that we would expect from a model without an intercept.

## (Q17)

The following plot shows the mean training and validation accuracies for 8-fold validation for different values of gamma (20 values sampled at logarithmic intervals in the range 0.01, 10):

We can see that our plots behave as expected according to the theory of underfitting/overfitting, where gamma is the relevant parameter.

For low values of gamma, we have underfitting, which means both training and validation accuracies are low since the model barely fits the data. This occurs since the values of are significant for many points , so many of ’s neighbors have a significant impact on the decision function (similarly to high k values in kNN).

Around gamma=1 we reach a peak (“sweet spot”), and as gamma continues to grow, training accuracy grows and validation accuracy decreases since the model is overfit to the training set and therefore does not generalize well to the validation set. This occurs since is significant only for ’s that are very close to , so very few of ’s neighbors have a significant impact on the decision function (similarly to low k values in kNN).

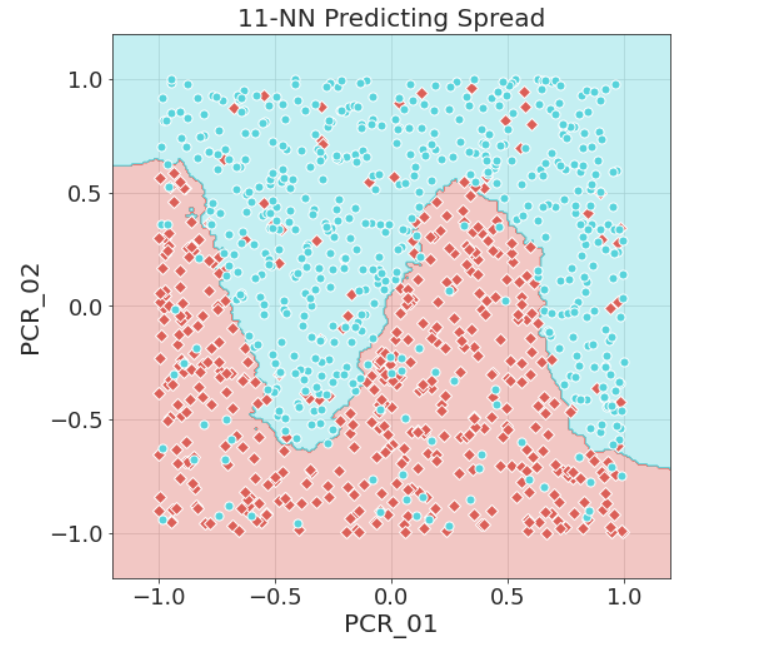
The optimal value of gamma (best validation accuracy) is gamma=1.623 with a mean training accuracy of 0.877 and a mean validation accuracy of 0.866.

Maybe choose a different gamma, if so maybe write this:

There are multiple values of gamma with almost identical validation accuracies, so we choose the gamma value closer to 0.1, since from the plots in (Q17) we can see that for gamma=1 the decision regions exhibit strange behavior. We chose gamma=0.127 with a mean training accuracy of 0.845 and a mean validation accuracy of 0.841.

## (Q18)

The following plot shows the decision regions for the optimal gamma we found, side by side with the optimal kNN model (11-NN):



spread:

1

-1

The test accuracy for the RBF kernel model is 0.872, while the test accuracy for the 11-NN model is 0.892. This means that based on the test accuracy alone, the 11-NN model is better for the task of predicting *spread* than the RBF model.

However we can notice some qualitative differences between the plots:

1. In general, the RBF model is “smoother”, meaning that near the edges of the boundary decisions, it is less overfit to the training samples compared to the “jagged” formation of the 11-NN model. This is an advantage of the RBF model since we assume (reasonably) that the underlying distribution is smooth.

2. The RBF model has strange behaviors of the decision regions near the edges of the feature plane, which is a disadvantage since outliers in the test set with edge feature values can be classified wrongly.