# Homework

**Course:** Introduction to Machine Learning

**Assignment:** Major HW2

**Submitters:** Ofer Nissim – 312367576

Lahav Fridlander – 209403781

**Date:** 22/12/22

## Part 1: Basic model selection with k-Nearest Neighbors

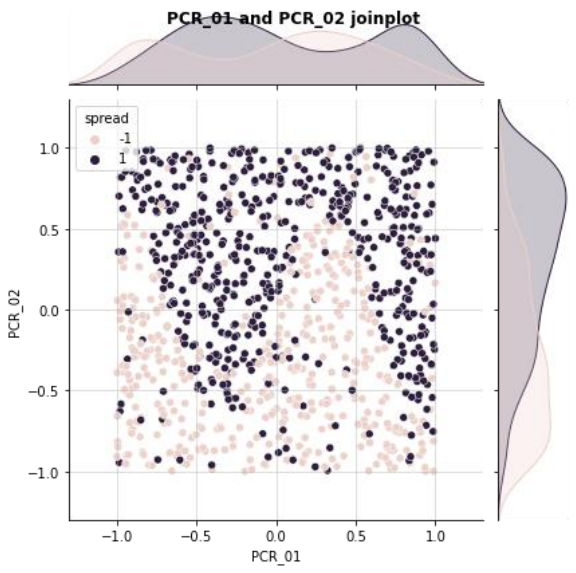
### Visualization and basic analysis

**Task**: Create a temporary `DataFrame` containing only `PCR_01` and `PCR_02`.

**Reminder**: Use the preprocessed (normalized) training set.

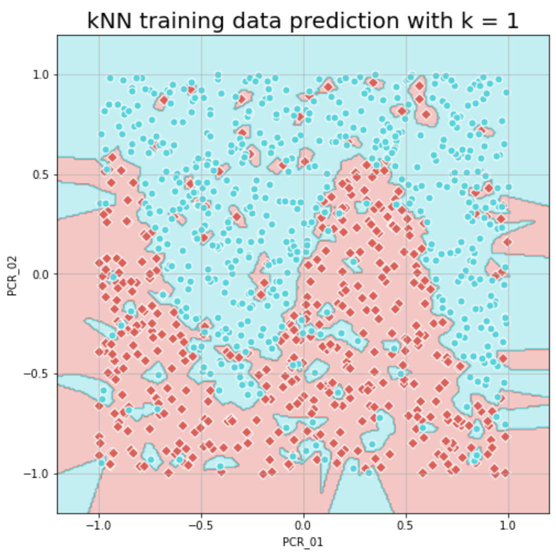
**(Q1)** Attach a `jointplot` of `PCR_01`, `PCR_02` (use the `spread` class as the `hue` argument).

**The obtained joint plot:**



**(Q2)** Train a k-NN model using $k = 1$ on your training set and use the `visualize_clf` method (given in this HW) to visualize the resulted decision regions. Make sure to have appropriate title and labels.

**k-NN (k=1) resulted decision regions:**



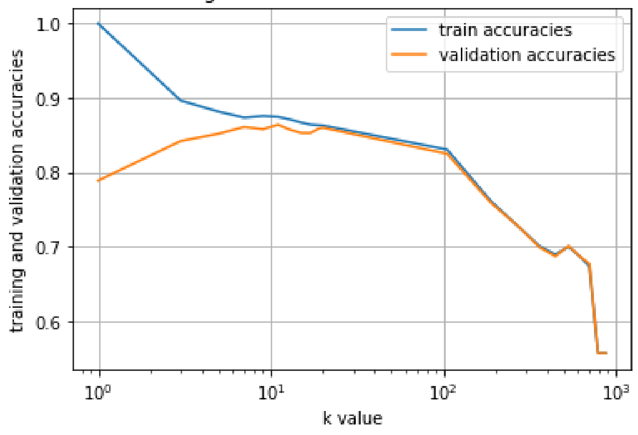
**(Q3)** Use sklearn.model_selection.cross_validate to find the best $k$ (neighbors) value in `list(range(1, 20, 2)) + list(range(20, 871, 85))` for predicting the `spread` class using `PCR_01` and `02`. Read the API carefully to understand how to extract train scores. Use the (default) accuracy metric and 8-folds to perform cross-validation.

Using the outputs of `cross_validate`, plot a *validation curve*, i.e., the (mean) training and validation accuracies (y-axis) as functions of the $k$ values (x-axis). Make the x-axis logarithmic (using `plt.semilogx`) and attach the plot (with the 2 curves) to your report.

**Answer:** Which $k$ value is the best? What are its mean training and validation accuracies (computed during the cross validation)? Also explain which $k$ values cause overfitting and underfitting (and why is that).

**The obtained validation curve:**

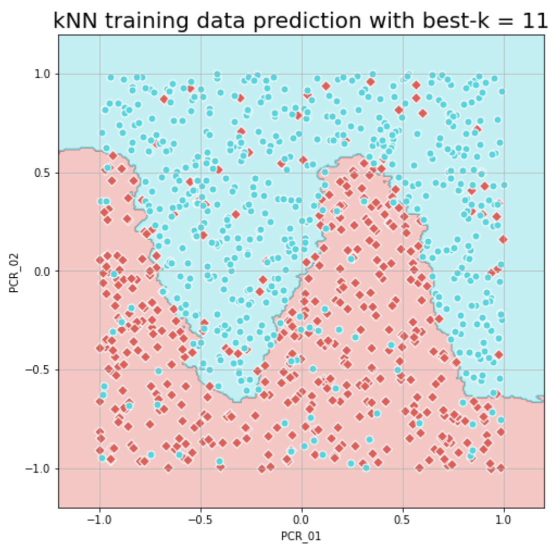
Cross-validation training and validation accuracies as a function of k-values



The k value that got the best cross-validation test score is **11**. Its mean training accuracy is 0.874 and its mean validation accuracy is 0.864. Low k-values (**1, 3**) cause overfitting, since they yield very high train accuracies in contrast to relatively low validation accuracies – which tells us that the model is fitted "too good" to the training data, which makes its performance worse for the "average" unknown data. On the other hand, high k-values (above 150) cause underfitting, as both training and validation accuracies are now lower – which indicates on incapability of the model to classify any relevant data correctly.

**(Q4)** Use the optimal $k$ value you found and retrain a k-NN model on <u>all</u> the training samples. In your report: plot the decision regions of this final model (using `visualize_clf`) and write its <u>test</u> accuracy (computed on the separate test split) of this model.

**<u>Decision regions of the final model:</u>**

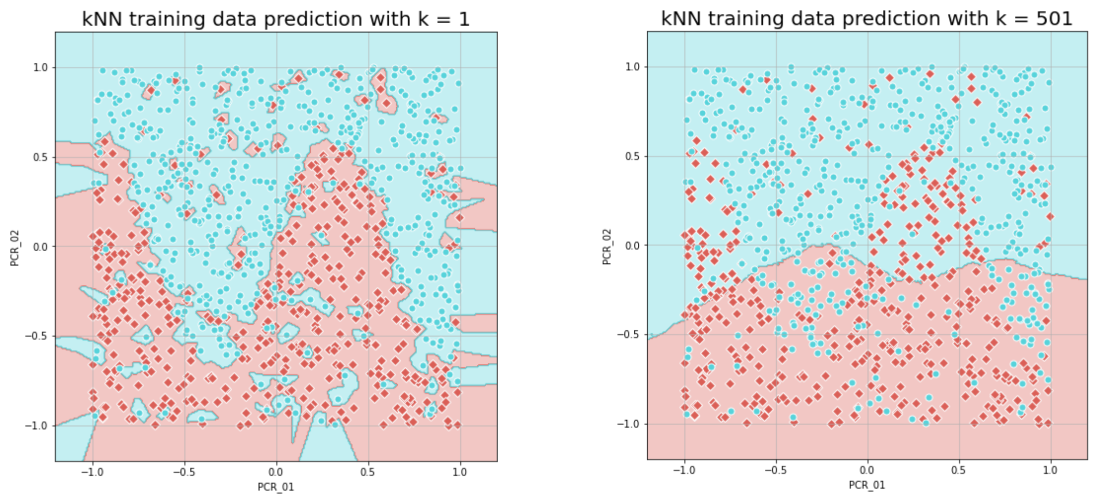


The test accuracy of this model is **0.896**.

**(Q5)** Using all training samples, train two additional models with $k = \{1, 501\}$. Visualize their decision regions as well.

Compare the boundaries of these two k-values to the ones of the optimal one you found earlier; and discuss the results and the exhibited behaviors (2-3 sentences).

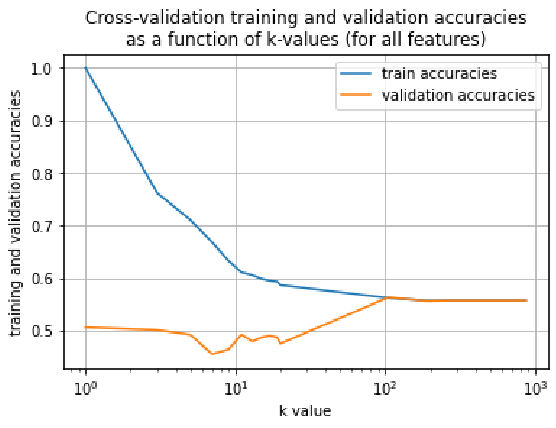
**<u>Decision region for $k = 1$ and $k = 501$:</u>**



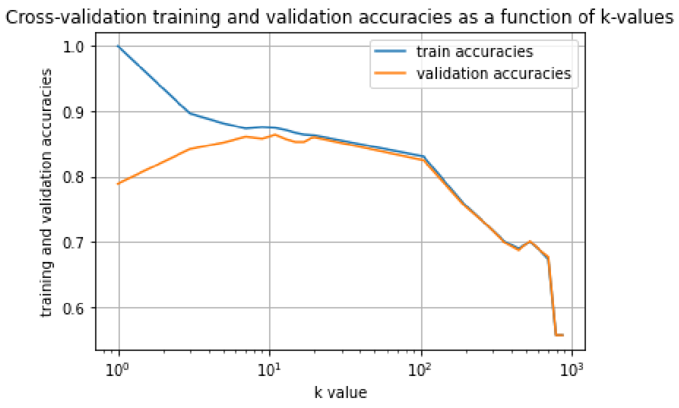
The boundaries of the $k = 1$ model are complex and considering a correct prediction to each sample, such that it seems like the model was "tailored specifically" to the training data – a situation that matches overfitting. The boundaries of the $k = 501$ model are forming a simple curve (with a linear-like trend) and are allowing a decent amount of prediction errors, a case that matches underfitting. By observing the boundaries of the $k = 11$ optimal model, we can tell that this model combines the attributes of both models – it exhibits a bit complex type of boundaries (sinus-like shape), but also allows a certain amount of prediction errors. As discussed in class, we understand that the optimal model expresses the trade-off between complex models, which have better accuracy on the training data but can also inflict overfitting, and simple models, that might classify the data more realistically on advance of prediction errors for both training are validation data, related to underfitting.

**(Q6)** Repeat (Q3) using all the features in the training dataset (instead of only `PCR_01` & `02`).
Add the validation curve (train and validation) to your report.
Discuss in detail the differences between the results here to those of (Q3) and try to explain them considering the mechanism of k-NN models.

**The obtained new validation curve:**



**The old validation curve (only for PCR_01 and PCR_02):**



The previous validation accuracy graph showed a classic underfitting to overfitting behavior- lower accuracies at the higher k values because of underfitting, then a best k value in the middle of the graph and overfitting lower k values at the end. The overfitting can be seen in the train to validation accuracy gap, while the underfitting is seen in the mutual curve drop.

Meanwhile, the new validation accuracy graph shows a similar behavior, but with lower validation accuracies for all k values. The overall accuracy difference can be attributed to the mechanism of k-NN models- we learned in class that k-NN doesn't perform well with many features, but rather prefers a low number of them. That stems from the primary influence of the distance between samples on the performance of the k-NN algorithm. Adding uninformative features to the k-NN algorithm, creates noise for the model to consider and results in lower performances.

The new graph also has a faster drop in the training accuracy graph. This behavior can be attributed to the k-NN lower performance with a higher feature amount (as stated above).

## Part 2: Decision trees

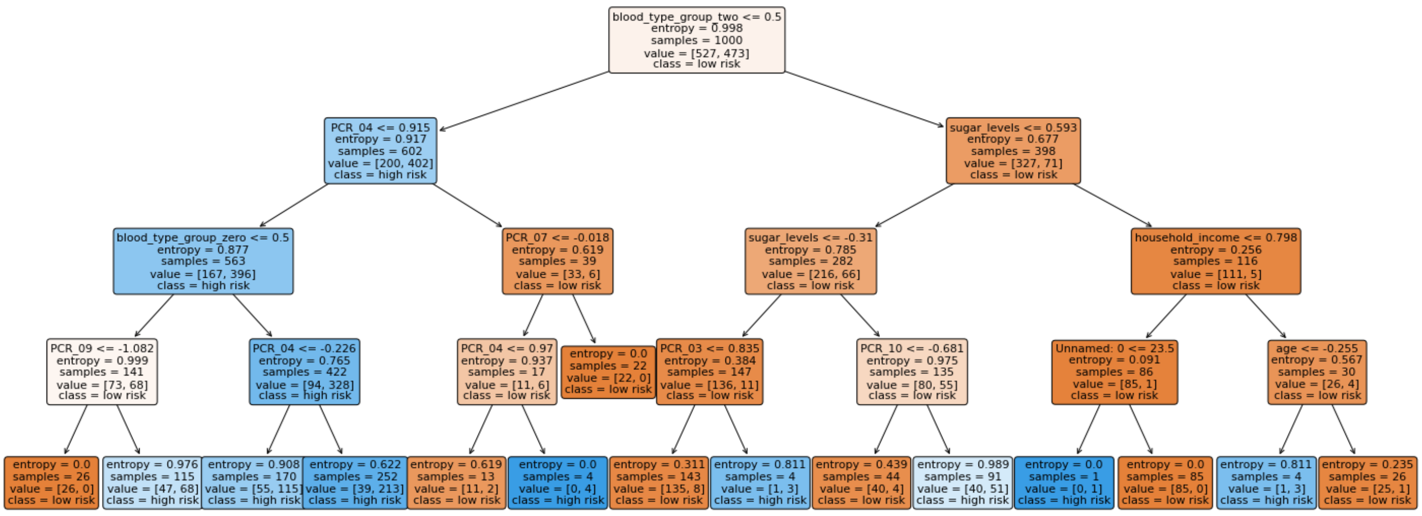
### Visualization

**(Q7)** Train a model with ID3 and `max_depth=4` (not including the root level; use the entire training set, i.e., all the features after preprocessing from all the training samples). What is the training accuracy?

Visualize the trained tree using `plot_tree` (provide feature and class names; use `filled=True`) and attach the plot to your report. The plot should be readable!

The obtained training accuracy is: **0.802**.

**Visualization of the trained tree:**

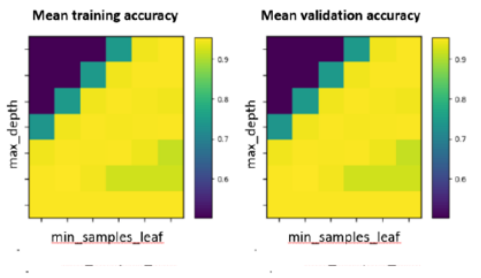

blood_type_group_two <= 0.5 entropy = 0.998 samples = 1000 value = [527, 473] class = low risk
PCR_04 <= 0.915 entropy = 0.917 samples = 602 value = [200, 402] class = high risk
sugar_levels <= 0.593 entropy = 0.677 samples = 398 value = [327, 71] class = low risk
blood_type_group_zero <= 0.5 entropy = 0.877 samples = 563 value = [167, 396] class = high risk
PCR_07 <= -0.018 entropy = 0.619 samples = 39 value = [33, 6] class = low risk
sugar_levels <= -0.31 entropy = 0.785 samples = 282 value = [216, 66] class = low risk
household_income <= 0.798 entropy = 0.256 samples = 116 value = [111, 5] class = low risk
PCR_09 <= -1.082 entropy = 0.999 samples = 141 value = [73, 68] class = low risk
PCR_04 <= -0.226 entropy = 0.765 samples = 422 value = [94, 328] class = high risk
PCR_04 <= 0.97 entropy = 0.937 samples = 17 value = [11, 6] class = low risk
entropy = 0.0 samples = 22 value = [22, 0] class = low risk
PCR_03 <= 0.835 entropy = 0.384 samples = 147 value = [136, 11] class = low risk
PCR_10 <= -0.681 entropy = 0.975 samples = 135 value = [80, 55] class = low risk
Unnamed: 0 <= 23.5 entropy = 0.091 samples = 86 value = [85, 1] class = low risk
age <= -0.255 entropy = 0.567 samples = 30 value = [26, 4] class = low risk
entropy = 0.0 samples = 26 value = [26, 0] class = low risk
entropy = 0.976 samples = 115 value = [47, 68] class = high risk
entropy = 0.908 samples = 170 value = [55, 115] class = high risk
entropy = 0.622 samples = 252 value = [39, 213] class = high risk
entropy = 0.619 samples = 13 value = [11, 2] class = low risk
entropy = 0.0 samples = 4 value = [0, 4] class = high risk
entropy = 0.311 samples = 143 value = [135, 8] class = low risk
entropy = 0.811 samples = 4 value = [1, 3] class = high risk
entropy = 0.439 samples = 44 value = [40, 4] class = low risk
entropy = 0.989 samples = 91 value = [40, 51] class = high risk
entropy = 0.0 samples = 1 value = [0, 1] class = high risk
entropy = 0.0 samples = 85 value = [85, 0] class = low risk
entropy = 0.811 samples = 4 value = [1, 3] class = high risk
entropy = 0.235 samples = 26 value = [25, 1] class = low risk

**(Q8)** Using 8-fold cross-validation, tune the two hyperparameters by performing a grid search (see GridSearchCV). Find the combination yielding the best validation error for predicting the `risk` class. You should:

a. Choose appropriate ranges for both hyperparameters. This may require a few attempts. To make things quicker when trying to find appropriate hyperparameter ranges, you can start by using only 2 folds.

After testing various values for the hyperparameters, the range we chose for max_depth is (2,20), focusing on the following values: 4,5,6,7,8,9,10,15,20. The range we chose for min_samples_leaf is (5,30), focusing on the following values: 2,5,10,15,20,25,30.

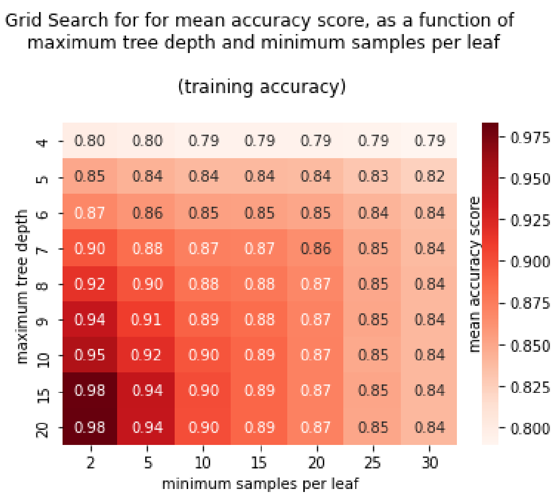
b. Since we tune two hyperparameters, instead of a validation curve, plot two heatmaps (seaborn / pyplot), one for the cross-validated training accuracy and one for the cross-validated validation accuracy.
These heatmaps should roughly have the following style / structure:

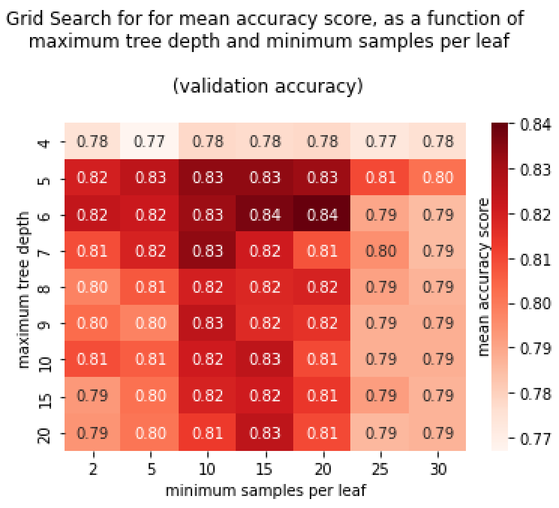


Make sure to plot the appropriate "ticks" on both axes and use annotations (`annot=True`) to explicitly write the accuracies inside the heatmap cells.
**Important:** The plots should be readable and informative!

c. Add the 2 plots to your report and specify which hyperparameter combination is optimal.

**The obtained heatmaps:**





As can be seen in the validation heatmap, the optimal hyperparameter combination is max_depth = 6 and min_samples_leaf = 20.

d. Write a hyperparameter-combination that causes underfitting.

A hyperparameter combination that causes underfitting is max_depth = 4 and min_samples_leaf = 30.

e. Write a hyperparameter-combination that causes overfitting.

A hyperparameter combination that causes overfitting is max_depth = 20 and min_samples_leaf = 2.

f. Add a short discussion regarding why each specific hyperparameter-combination from sub-questions 'd' and 'e' resulted in under/over-fitting.

We'll notice that the hyperparameter combinations from the previous clauses have extreme values within the ranges – the combination mentioned in sub-question 'd' has the minimal max depth and the maximal number of samples per leaf, whereas the combination mentioned in sub-question 'e' has the maximal depth and the minimal number of samples per leaf.

The first resulted in underfitting, since small tree depth limits the classifier's complexity and large number of samples per leaf allows lower precision for each decision region. These attributes form a 'rough' classification, which doesn't generalize the data well.

The second resulted in overfitting, because large tree depth provides high model complexity, and small number of samples per leaf cause smaller decision regions, which can excellently fit the training data. These properties contribute to a 'specific tailored' classifier, which fits the data too good and also doesn't generalize well.

**(Q9)** Use the optimal hyperparameter combination you found and retrain a decision tree on all the training samples. In your report write the test accuracy of this model.

The obtained retrained tree accuracy is: **0.852**.

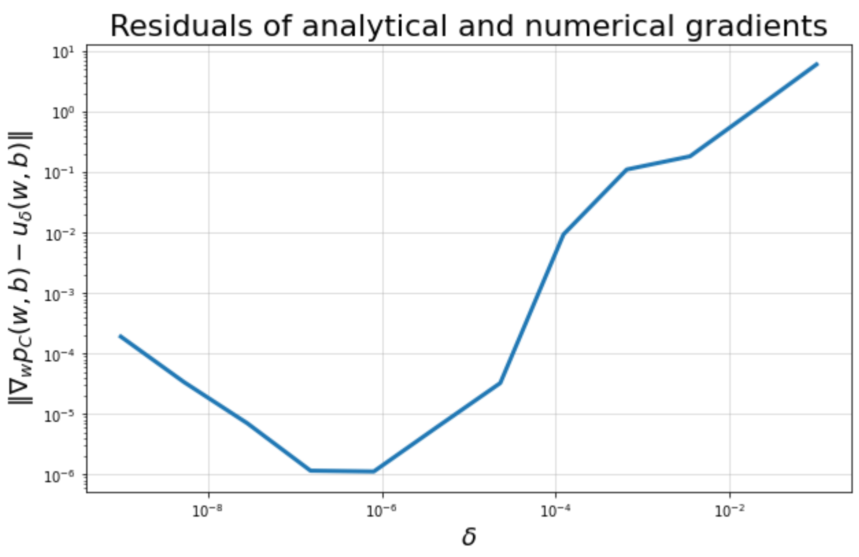
## Part 3: Linear SVM and the Polynomial kernel

**(Q10)** Using `PCR_01`, `PCR_02`, generate a plot that compares the numerical gradients to the analytic gradients. Do this by running the following command:

```
compare_gradients(X_train, y_train, deltas=np.logspace(-9, -1, 12))
```

Attach the plot to your report. Briefly discuss and justify the demonstrated behavior.

**The obtained comparison plot:**



The curve exhibits an expected behavior for $\delta \geq 10^{-6}$, which is monotonic increasing. This behavior makes sense, due to the definition of the derivative as a limit (as $\delta$ goes to zero, the numerical gradient becomes more similar to the analytical gradient, which implies for the residual getting smaller). For lower values, i.e., $\delta < 10^{-6}$, we observe a surprising behavior – the residual decreases as $\delta$ grows. An explanation for that behavior lies in the limitations of the numerical methods in computers, such that numerical instability is affecting the residual's value for very small deltas. This effect could turn over the curve's trend and demonstrate the aforementioned results.

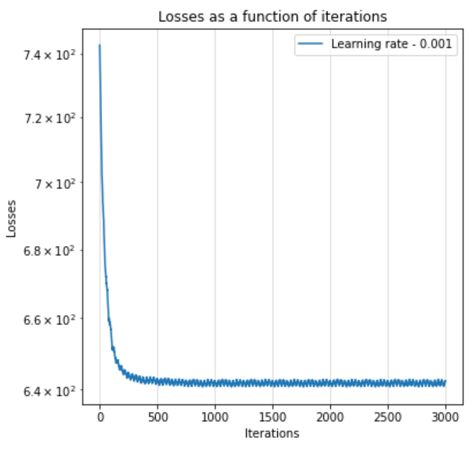
**(Q11)** Add the plot(s) to your report.

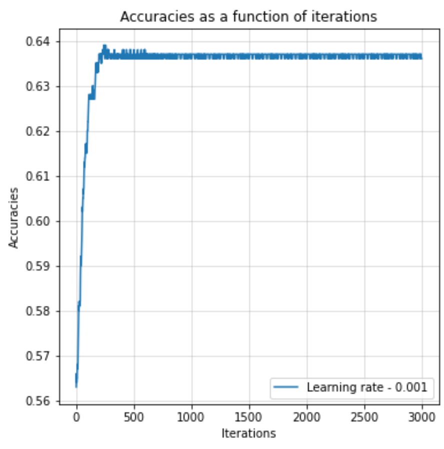
a. Given the plots, which learning rate would you choose?

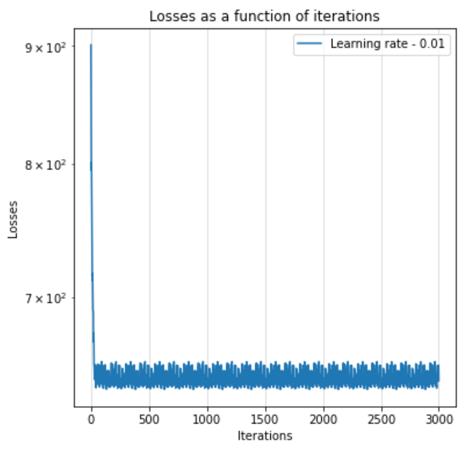
b. Train this linear model with the chosen learning rate and plot its decision regions (at the end of training).

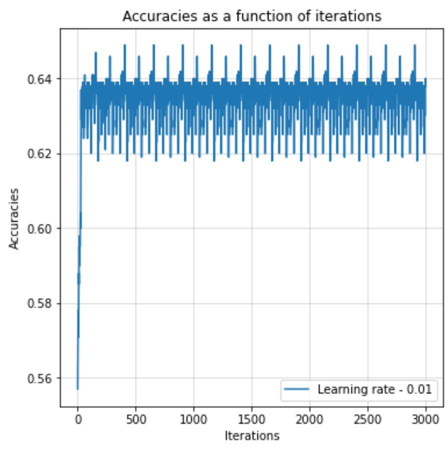
c. For the chosen learning rate, what is the maximal training accuracy and the minimal training loss achieved by your model (during training, according to your plots)? Are they attained at the same step? If so – must it be so? If not – how is it possible?

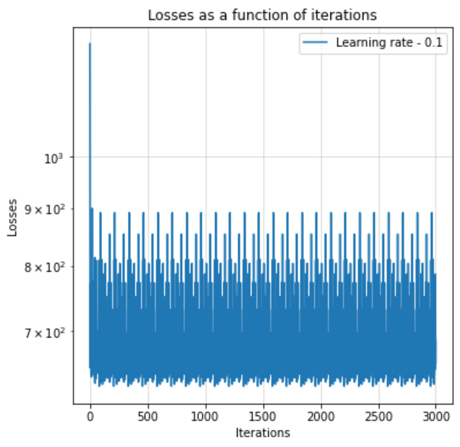
**<u>The obtained plots:</u> (with learning rates: 0.001, 0.01, 0.1, 1 – from top to bottom)**

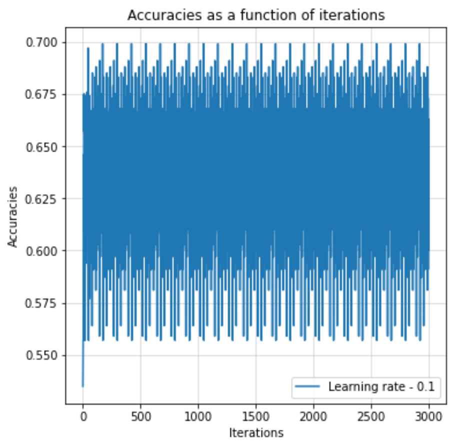


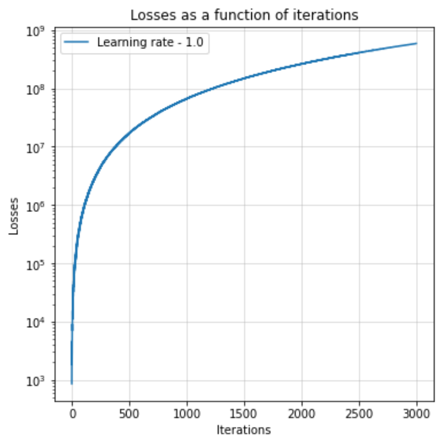


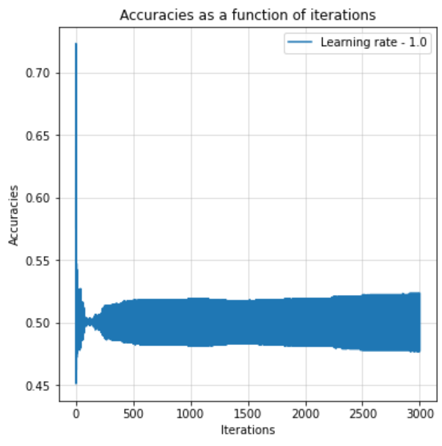








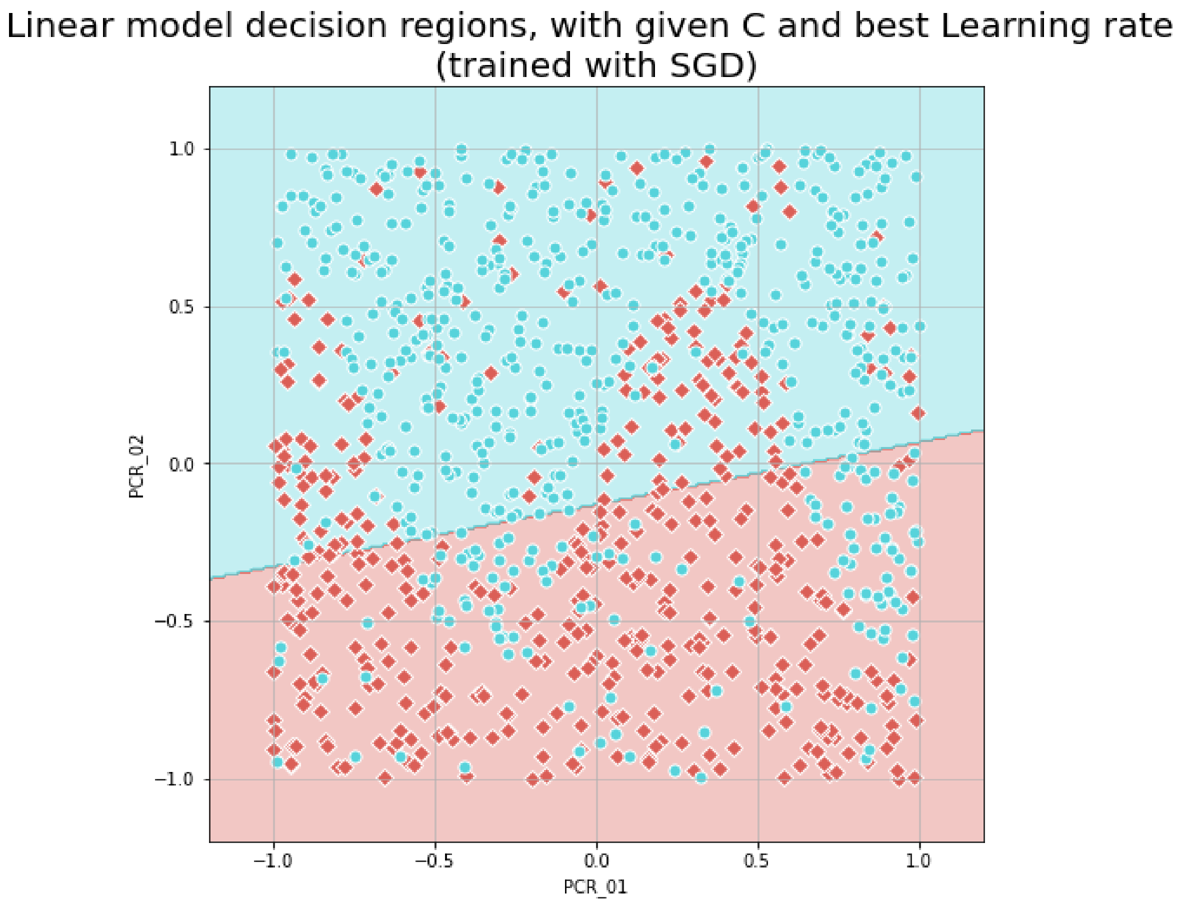




- As requested – we'll mention that we **modified the value of $C$ to 1** (instead of 0.1), due to convergence to W vector which does not improve the accuracy (lower C value causes too much emphasis on complexity, instead of the margin sum).

a. Given the plots, we chose the 0.01 learning rate. This learning rate gives a high accuracy, with a small amount of iterations. It's also not as noisy as the higher learning rate 0.1. His loss converges, unlike the 1 learning rate. The 0.01 learning rate is less stable then the 0.001 learning rate, but his accuracy is still high as the 0.001 accuracy and he converges much faster (less than 100, while the smaller learning rate takes about 500 iterations).

b.



C. **Maximal training accuracy:** around 0.65

**Minimal training loss:** around 650.

They aren't attained at the same step. This phenomenon is possible because the loss function doesn't give an exact criteria for better accuracies. The soft SVM loss is a function that balances between the model complexity and its margins.
If we had a C that approaches infinite value, we would only consider the margins, and strive only for better accuracy. But high C values tend to overfit, and thus we prefer to consider the models complexity in its loss calculation.

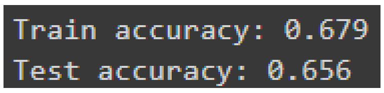
**(Q12)** In your report:

(a) Write the respective train and test accuracies of both models (2nd and 3rd degree).

(b) Add a plot of the decision boundaries for each classifier.

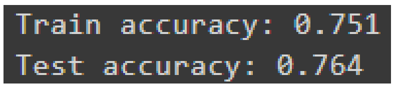
(c) Briefly discuss and explain the difference in decision regions between the two models.

(a) Accuracy results for 2nd degree model:



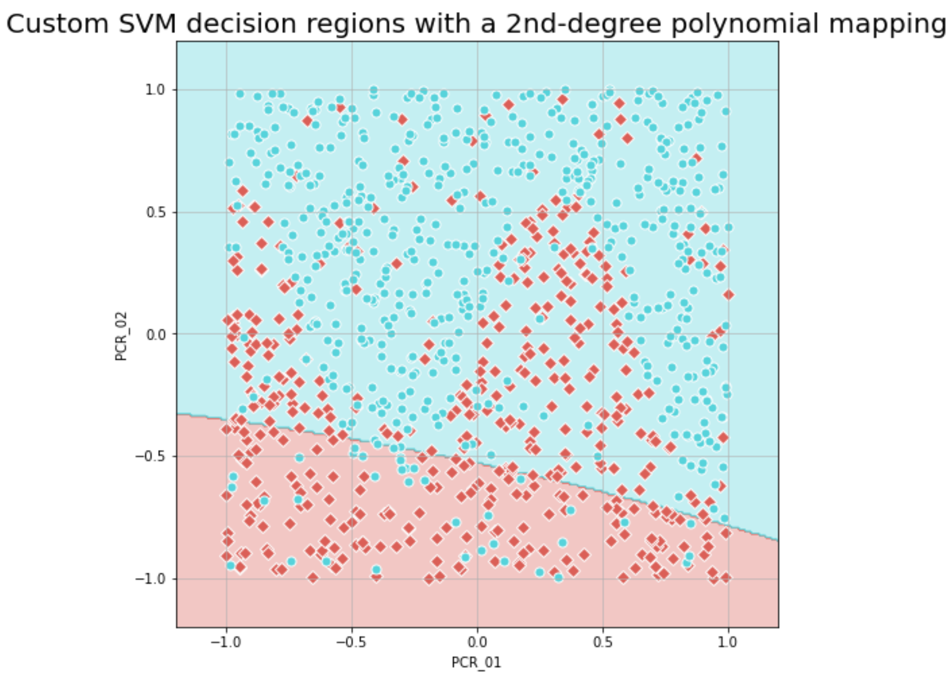
```
Train accuracy: 0.679
Test accuracy: 0.656
```

Accuracy results for 3rd degree model:



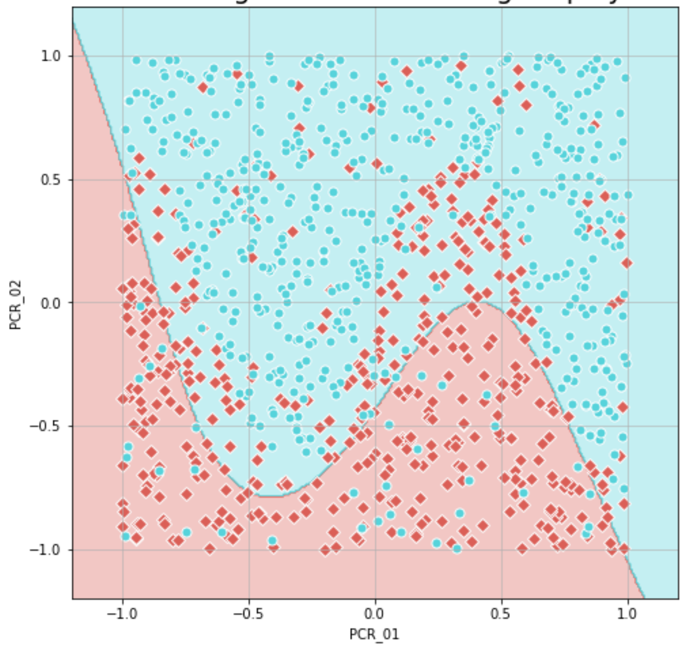
```
Train accuracy: 0.751
Test accuracy: 0.764
```

(b) <u>**2nd degree polynomial mapping resulted decision regions:**</u>



**3rd degree polynomial mapping resulted decision regions:**

Custom SVM decision regions with a 3rd-degree polynomial mapping



(c) The 2nd degree polynomial decision region is almost linear, while the 3rd degree polynomial is more suited to the visually seen data regions. This difference can be attributed to the different polynomial degrees- a 3rd degree polynomial has more flexibility (than 2nd degree) and therefore the mapping is more diverse. Meaning the mapping can suit the data better.

**(Q13)** Train the 3rd degree polynomial mapping SVM 5 times (on all the training samples), using the hyperparameters written in the snippet above.

In your report:

(a) Write the 5 resulting train accuracies, their mean, and their standard deviation.

(b) Plot the 5 resulting decision regions (use `visualize_clf` on the training set). Remember that you are evaluated on the aesthetics of your report as well.

(c) Answer: what is (are) the source(s) of the variability in the resulting models? Your answer should refer also to the convexity of the problem.

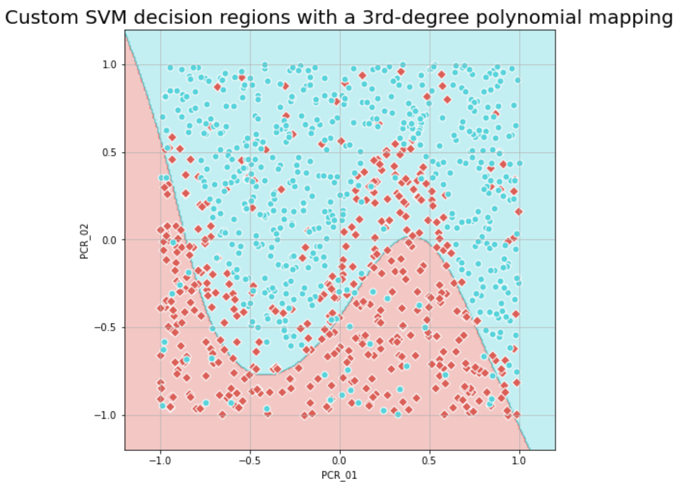
(a)

| Iteration | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Train accuracy | 0.752 | 0.751 | 0.756 | 0.749 | 0.755 |

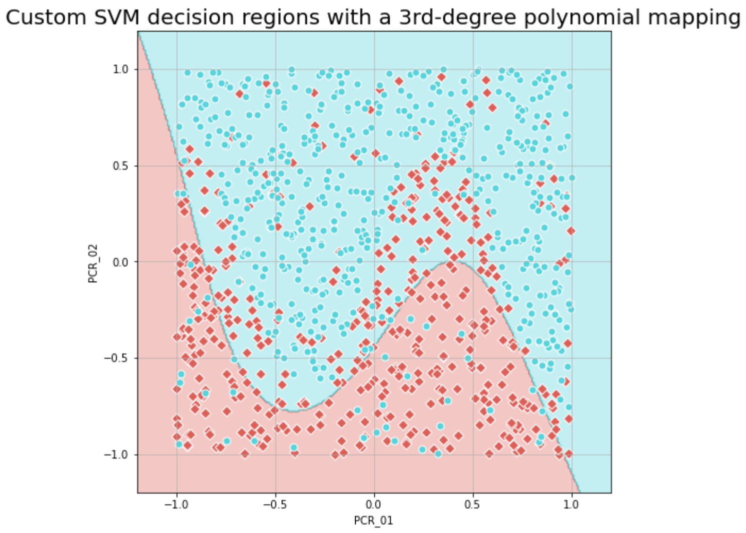
**The mean of the train accuracy is:** 0.7526

**The standard deviation of the train accuracy is:** 0.00257682

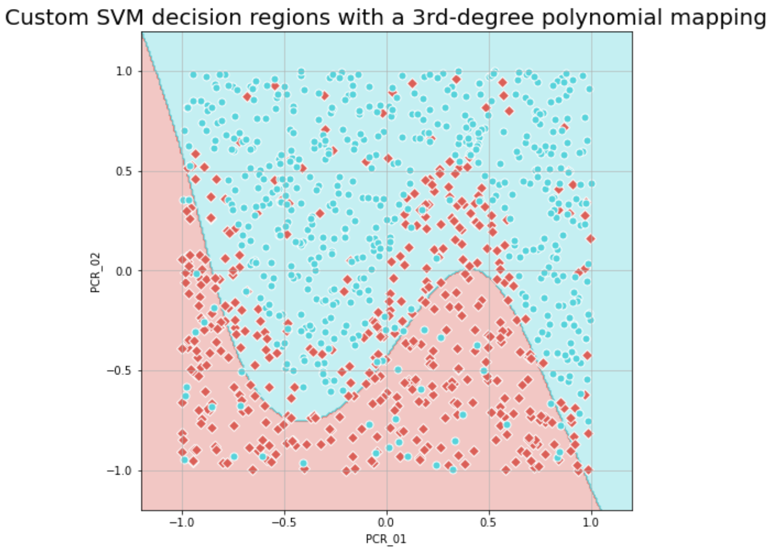
(b) **3rd degree polynomial mapping resulted decision regions (first iteration):**



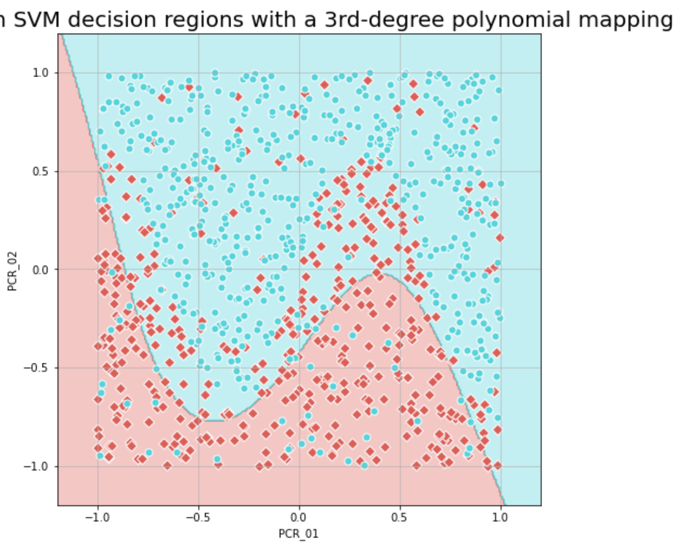
**3rd degree polynomial mapping resulted decision regions (second iteration):**



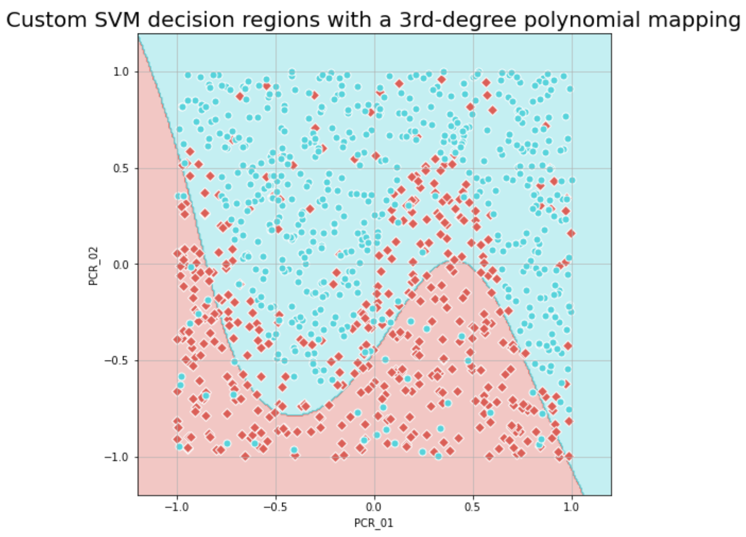
**3rd degree polynomial mapping resulted decision regions (third iteration):**



**3rd degree polynomial mapping resulted decision regions (fourth iteration):**



**3rd degree polynomial mapping resulted decision regions (fifth iteration):**



(c) The variability in the resulting models comes from the random choices in SGD- we start with a different W vector in each iteration and proceed while using different batches of samples. As we can see, the results are quite similar, because of the loss function convexity. But there's still a difference, as the results don't reach the exact minimum, and approach the minimum from different directions.

## Part 4: The RBF kernel

**(Q14)** Complete the derivation above to prove the behavior of RBF on the extreme of $\gamma \to \infty$.

That is, prove that:

$\lim_{\gamma\to\infty} \operatorname{sign}\left(\sum_{i\in[m],\alpha_i>0} \alpha_i y_i \exp\{-\gamma\|\boldsymbol{x}-\boldsymbol{x}_i\|_2^2\}\right) = y_{i^*}$, *where* $i^* = \operatorname{argmin}_{i\in[m],\alpha_i>0}\|\boldsymbol{x}-\boldsymbol{x}_i\|_2^2$.

<u>Simplification</u>: you can assume that no two samples $i \neq j$ hold $\|\boldsymbol{x}-\boldsymbol{x}_i\| = \|\boldsymbol{x}-\boldsymbol{x}_j\|$.

<u>Hint</u>: $\forall t \in \mathbb{R}, \alpha \in \mathbb{R}_{>0}$: $\operatorname{sign}(t) = \operatorname{sign}\left({}^{t}/_{\alpha}\right)$.

**<u>Our proof (exported from Lyx):</u>**

From the simplification, exists a single $i^*$, where $i^* = \underset{i\in[m],\alpha_i>0}{argmin}||x-x_i||_2^2$.

Without loss of generality, we'll assume that $y_{i^*} = 1$.

$$\sum_{i\in[m],\alpha_i>0} \alpha_i y_i e^{-\gamma\cdot||x-x_i||_2^2} \geq \sum_{i\in[m],\alpha_i>0,y_i=1} \alpha_i y_i e^{-\gamma\cdot||x-x_i||_2^2} + \sum_{i\in[m],\alpha_i>0,y_i=-1} \alpha_i y_i e^{-\gamma\cdot||x-x_i||_2^2} =$$
$$= \sum_{i\in[m],\alpha_i>0} \alpha_i e^{-\gamma\cdot||x-x_i||_2^2} - \sum_{i\in[m],\alpha_i>0} \alpha_i e^{-\gamma\cdot||x-x_i||_2^2} \ (1)$$

We'll define $j = \underset{i\in[m],y_i=-1}{argmax}\ \alpha_i, k = \underset{i\in[m],\alpha_i>0,y_i=-1}{argmin}||x-x_i||_2^2$.

And we denote $t$ as the number of $i \in [m], \alpha_i > 0$, such that $y_i = -1$.

$$(1) \sum_{i\in[m],\alpha_i>0} \alpha_i e^{-\gamma\cdot||x-x_i||_2^2} - \sum_{i\in[m],\alpha_i>0} \alpha_i e^{-\gamma\cdot||x-x_i||_2^2} \geq$$
$$\geq \sum_{i\in[m],\alpha_i>0} \alpha_i e^{-\gamma\cdot||x-x_i||_2^2} - \alpha_j \cdot \sum_{i\in[m],\alpha_i>0} e^{-\gamma\cdot||x-x_i||_2^2} \geq$$
$$\geq \sum_{i\in[m],\alpha_i>0} \alpha_i e^{-\gamma\cdot||x-x_i||_2^2} - \alpha_j \cdot \sum_{i\in[m],\alpha_i>0} e^{-\gamma\cdot||x-x_k||_2^2} =$$
$$= \sum_{i\in[m],\alpha_i>0} \alpha_i e^{-\gamma\cdot||x-x_i||_2^2} - \alpha_j \cdot t \cdot e^{-\gamma\cdot||x-x_k||_2^2} \geq$$
$$\geq \alpha_{i^*} e^{-\gamma\cdot||x-x_{i^*}||_2^2} - \alpha_j \cdot t \cdot e^{-\gamma\cdot||x-x_k||_2^2}$$

Note that $||x-x_{i^*}||_2^2 < ||x-x_k||_2^2$.

$$\lim_{\gamma\to\infty}\left(\frac{\alpha_{i^*}\cdot e^{-\gamma\cdot||x-x_{i^*}||_2^2}}{\alpha_j\cdot t\cdot e^{-\gamma\cdot||x-x_k||_2^2}}\right) = \lim_{\gamma\to\infty}\left(\frac{\alpha_{i^*}}{\alpha_j\cdot t}\cdot e^{-\gamma\cdot(||x-x_{i^*}||_2^2-||x-x_k||_2^2)}\right) =$$
$$= \lim_{\gamma\to\infty}\left(\frac{\alpha_{i^*}}{\alpha_j\cdot t}\cdot e^{\gamma\cdot(||x-x_k||_2^2-||x-x_{i^*}||_2^2)}\right) \underset{||x-x_k||_2^2-||x-x_{i^*}||_2^2>0}{=}$$
$$= \infty$$

Therefore, exists $n \in \mathbb{R}$ such that for every $x > n : \frac{\alpha_{i^*}\cdot e^{-\gamma\cdot||x-x_{i^*}||_2^2}}{\alpha_j\cdot t\cdot e^{-\gamma\cdot||x-x_k||_2^2}} > 1 \Rightarrow$

$\Rightarrow \alpha_{i^*}\cdot e^{-\gamma\cdot||x-x_{i^*}||_2^2} > \alpha_j \cdot t \cdot e^{-\gamma\cdot||x-x_k||_2^2}$

Meaning that for every $x > n : \alpha_{i^*} e^{-\gamma\cdot||x-x_{i^*}||_2^2} - \alpha_j \cdot t \cdot e^{-\gamma\cdot||x-x_k||_2^2} > 0 \Rightarrow$

$$\Rightarrow \forall x > n : \sum_{i\in[m],\alpha_i>0} \alpha_i y_i e^{-\gamma\cdot||x-x_i||_2^2} \geq$$
$$\geq \alpha_{i^*} e^{-\gamma\cdot||x-x_{i^*}||_2^2} - \alpha_j \cdot t \cdot e^{-\gamma\cdot||x-x_k||_2^2} > 0 \Rightarrow$$

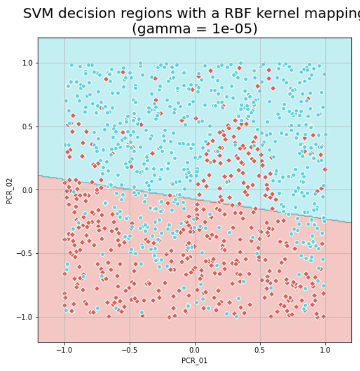
$$\Rightarrow \forall x > n : sign\left(\sum_{i\in[m],\alpha_i>0} \alpha_i y_i e^{-\gamma\cdot||x-x_i||_2^2}\right) = 1 \Rightarrow$$

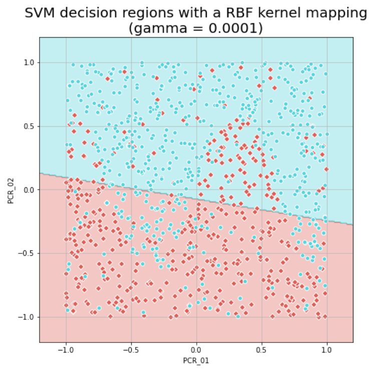
$$\Rightarrow \lim_{\gamma\to\infty} sign\left(\sum_{i\in[m],\alpha_i>0} \alpha_i y_i e^{-\gamma\cdot||x-x_i||_2^2}\right) = 1 = y_{i^*}$$

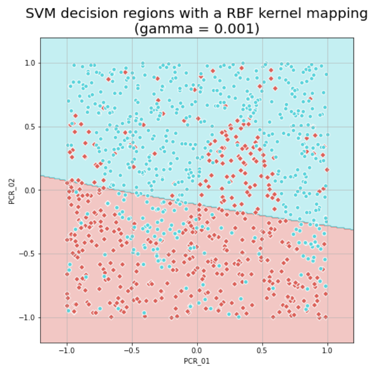
**(Q15)** Use our visualization method to visualize the ten models' decision regions (use the `marker_size` argument to adjust the plots for small decision regions).

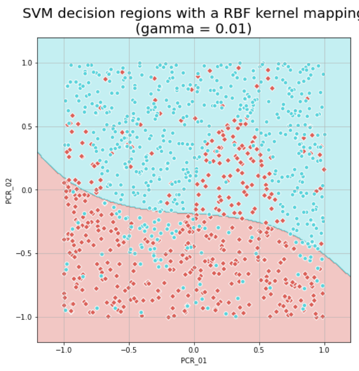
Attach the ten plots to your report (resize them to fit in a single page of your report). Remember that all plots should have suitable titles. You are evaluated on the aesthetics of your report as well.

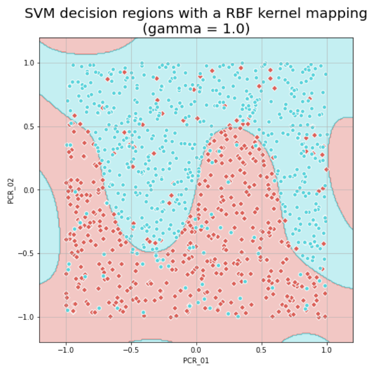
**The obtained models' decision regions: (for $\gamma = 10^i$, where $i \in \{-5, -4, \dots, 3, 4\}$ by ascending order):**

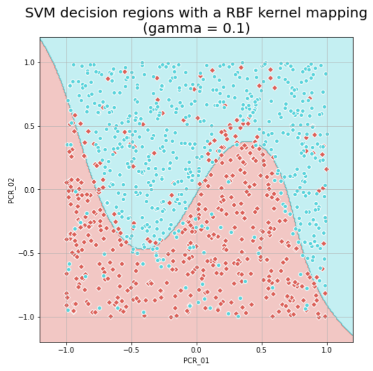


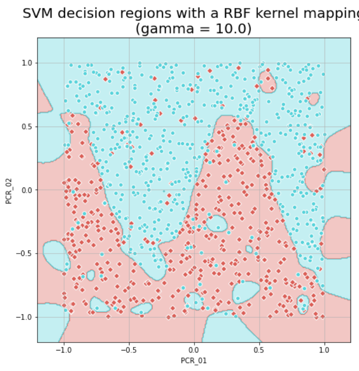


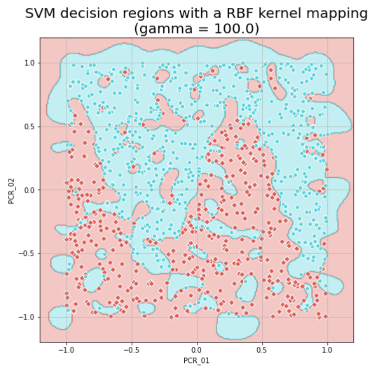


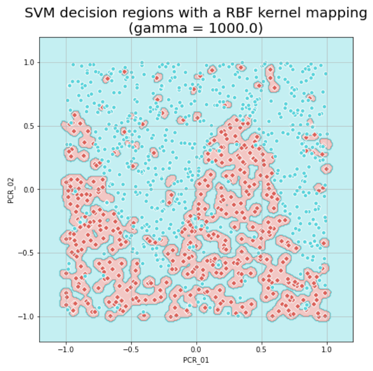


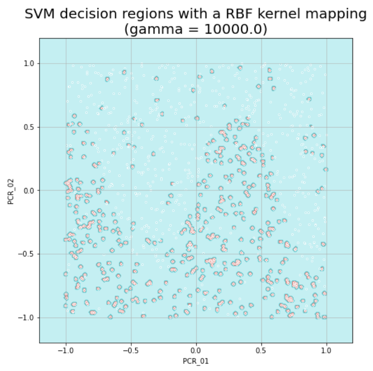




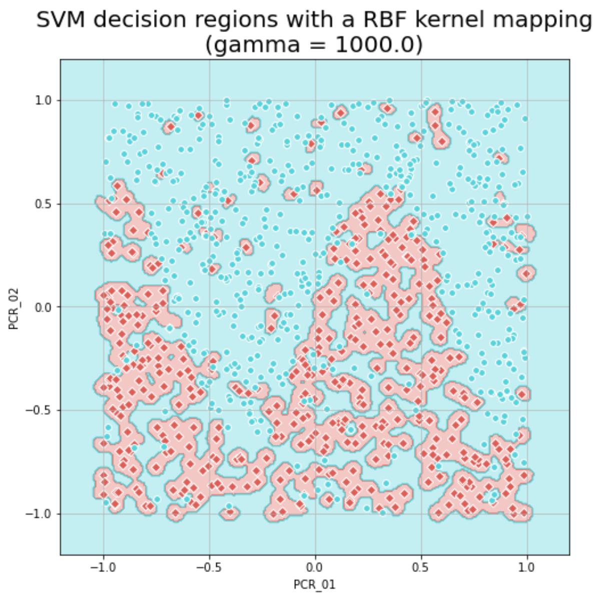


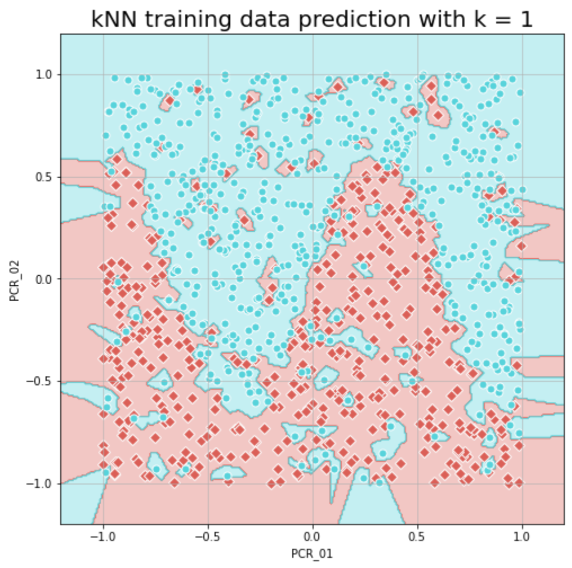






**(Q16)** Compare the decision regions of the k-NN model with $k = 1$ from (Q2) to those of the RBF model with $\gamma = 10^4$ from the previous question. We expected the two models to be very similar, but we should still see significant differences. Try to explain the differences (try to give several reasons, there isn't a single correct answer).





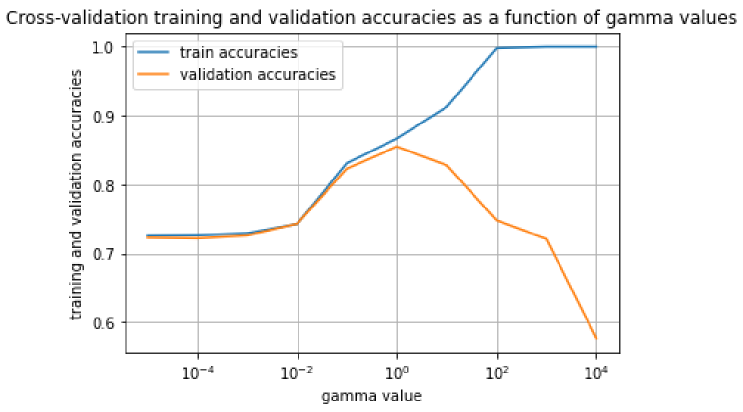
Both the RBF and the k-NN models create a complete separation of the data points, thus creating overfitted decision regions.
But the k-NN model marks territories according to the most approximate point. Therefore, the separation lines are in the middle of different labeled data points.
The RBF model seems to default to the blue label, and only creates red territories near the red data points. This could be caused by the operating method of the RBF model - using greater gamma values means that each point has a shorter reach. Therefore, with a large enough gamma value, we'll have middle grounds that no point reaches them (their numerical influence over the calculation of the points prediction is zero). Those area might default to blue territory.

**(Q17)** Now, given the problem at hand (predicting the `spread` using an SVM with the RBF kernel on `PCR_01` and `02`), we wish to find an optimal $\gamma$ given `C=1e4`. To this end, perform 8-fold cross validation on $\gamma$.

Remember: when looking for appropriate value intervals for $\gamma$, you may start by using only two folds. Once the ranges are set, increase the folds' number to 8 and sample many values throughout the interval. We recommend sampling the hyperparameter values at logarithmic intervals for good results.

Attach here the required plots and explanations, as in (Q8).

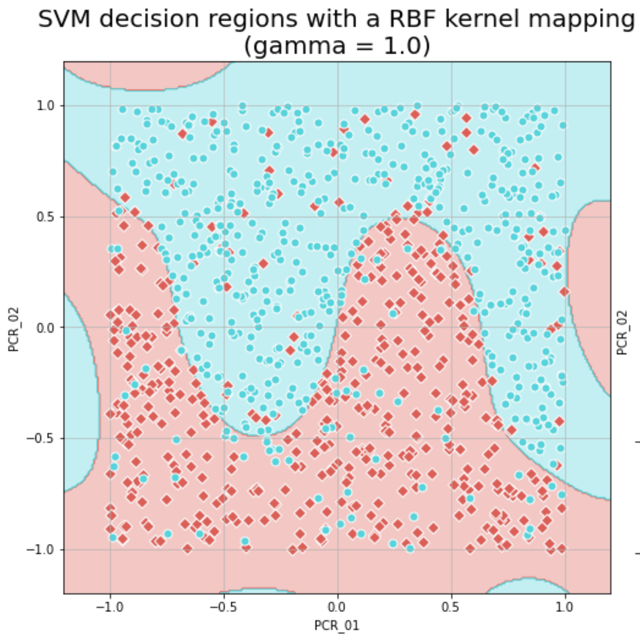
**The obtained validation curve:**

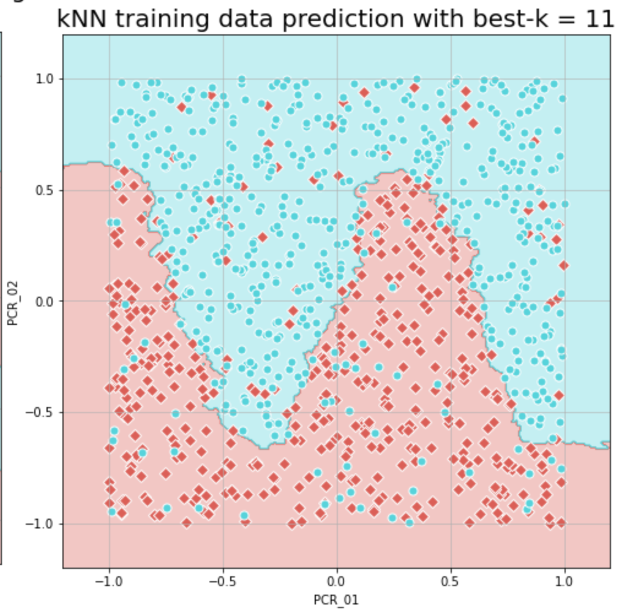


The $\gamma$ value that got the best cross-validation test score is **1**.
Its mean training accuracy is 0.867 and its mean validation accuracy is 0.855.
Similar to our answer in Q3 (where $\gamma$ has the inverse effect of k) - High $\gamma$-values (**10, 100, 1000, 10000**) cause overfitting, since they yield very high train accuracies in contrast to relatively low validation accuracies – which tells us that the model is fitted "too good" to the training data, which makes its performance worse for the "average" unknown data. On the other hand, low $\gamma$ -values (above 150) cause underfitting, as both training and validation accuracies are now lower – which indicates on incapability of the model to classify any relevant data correctly.

**(Q18)** Use the optimal hyperparameter you found and retrain an RBF model on the all the training samples. In your report: plot the decision regions of this final model and write its test accuracy. Compare these results to those from (Q4) in 3-4 sentences.
Based on the test accuracy, which model is better for the task of predicting the `spread`?

**Decision regions of the RBF model using optimal $\gamma$:** **kNN decision regions using the optimal k:**





The results of the RBF model are similar to the k-NN results from Q4, as the general shapes of both decision regions are alike. A noticeable difference is the "islands" formed around the edges of the regions of the RBF model, which stem from the attributes of the RBF kernel method, as mentioned earlier (in Question 16).
Another difference is the "smooth" transition between regions of the RBF in comparison to the k-NN model. This property relies on the fact that the RBF considers all the nearest points in the environment (as it's a linear combination of Gaussians, with width determined by $\sigma$), whereas k-NN takes into account only a fixed (discrete) number of points.

The test accuracy of this model is **0.9**. Based on the test accuracy (the kNN model accuracy is 0.896 in Q4), the **RBF MODEL** is better for the task of predicting the spread.