

HW2: Algorithm Implementation and Basic Model Selection

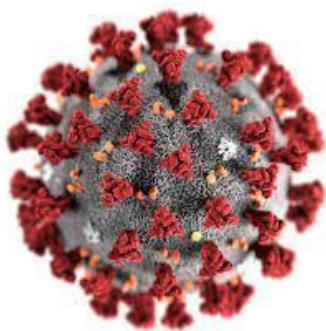
Goal

This assignment is the second of three mini projects that will guide you in your task of stopping the spread of disease around the globe!

In this assignment you will implement Soft-SVM using gradient descent. You will also practice basic hyperparameter tuning (model selection) using three algorithms: k-NN, Decision Trees (with ID3), and Soft-SVM.

Many techniques and ideas from this assignment will greatly help you in Major HW3.

Good Luck!



Instructions

- **Submission**
 - Submit by: Thursday, 25.07.2024, 23:59.
- **Python environments and more**
 - We recommend using jupyter notebooks. [Google colab](#) can be very convenient since it does not require installing anything on your local computer. It will also help you to collaborate with your partner online.
 - Initial notebook [here](#).
 - Demonstrates how to upload a dataset to Google colab and how to download files from Google colab.
 - You can save a copy of this notebook to your Google drive.
 - However, you can use any Python IDE you choose. For working locally with an IDE, we recommend first installing [conda](#) for package management (with Python 3.6 or 3.8), and then installing an IDE like [PyCharm](#) or [Spyder](#).
- **Your code**
 - Should be clearly and briefly documented.
 - Variables/classes/functions should have meaningful names.
 - Will be partially reviewed and graded.
- **Final report**
 - Should be written in a word processor (Office Word, Google docs, etc.).
 - Should not contain the code itself.
 - Do not submit jupyter notebooks as PDFs.
 - Can be in Hebrew, English, or both.
 - **You are primarily assessed based on your written report.**
 - Answer the questions in this instruction file according to their numbering.
 - Add concise explanations, figures (outputs of your code), tables, etc.
 - You are evaluated for your answers but also for clarity, readability, and aesthetics.
 - **Tables** should include feature names and suitable titles.
 - **Plots** should have suitable titles, axis labels, legends, and grid lines (when applicable)
 - We recommend adjusting the default font sizes of matplotlib (see snippet in HW1).

- **Submit a zip file containing** (please use hyphens, not underscores):
 - Define `<filename>` as your dash-separated IDs, i.e., `id1-id2`.
 - The zip file's name should be `<filename>.zip` (e.g., `123456789-200002211.zip`).
 - **Only one group member should submit the assignment to the webcourse!**
 - The report PDF file with all your answers (but not your code!), named `<filename>.pdf`.
 - Your code:
 - The following files (separately, regardless of whether they appear in the main notebook):
 - `SoftSVM.py`: your completed SVM module (=class).
 - Also:
 - Working with jupyter: your notebooks, `<filename>.ipynb`.
 - Working with a “traditional” IDE: one clear main script, `<filename>.py`, and any additional files required for running the main script.
 - Do not submit csv files.

● **Failing to follow any of the instructions above may lead to point deduction!**

Preliminary: Data Loading

Task: Follow the procedure below.

- a. Start by **loading** the preprocessed data from the previous assignment.

Note: in Lecture 08 we explain why some preprocess steps (e.g., normalization), should be applied to the validation folds according to statistics computed on the train fold.

Here, for simplicity only, you compute these statistics according to the all the training samples (after splitting to train-test, before splitting the training set to additional folds).

- b. Make sure the data is **partitioned** correctly to train and test, according to the instructions in the previous assignment.

The train-test partitions **must** be identical to the ones you used in HW1.

- c. Following questions that we received: You shouldn't have deleted features in HW1 other than perhaps `patient_id`, `PCR_date`, and `current_location`, since it was too early for you to decide whether they are useful for classification. If you deleted any other features, that is fine, but now you should bring them back and edit your preprocessing pipeline accordingly (including suitable normalization steps).

- d. Make sure target variables follow the `{+1, -1}` convention (rather than `{1, 0}` or `{True, False}`).

Part 1: Basic model selection with k-Nearest Neighbors

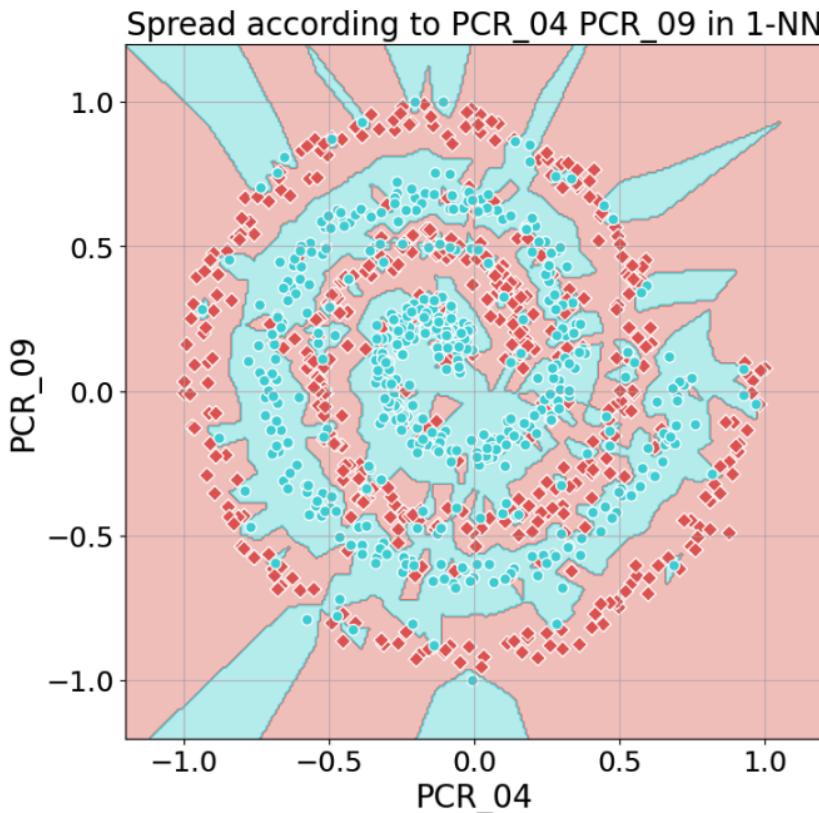
Like in HW1, we start with the simple k-NN model and use it to practice new concepts, like model selection. Here we use k-NN on `PCR_04` and `PCR_09` to predict the `spread`, similarly to HW1. Use `sklearn.neighbors.KNeighborsClassifier` rather than your custom implementation from HW1.

Visualization and basic analysis

Task: Create a temporary `DataFrame` containing only `PCR_04` and `PCR_09`.

Reminder: Use the preprocessed (normalized) training set.

(Q1) Train a k-NN model using $k = 1$ on your training set and use the `visualize_clf` method to visualize the resulted decision regions (with appropriate title and labels).



Model selection

Most ML models are characterized by a set of parameters that control the learning process and are not optimized during training itself (e.g., k in k-NN or C in SVM). These hyperparameters can change the final model dramatically. Hyperparameters are often tuned using [k-fold-cross-validation](#), where we split the training set into k_v folds and train k_v models – each trained on $k_v - 1$ folds and validated on the remaining one. This procedure estimates

the model's performance on unseen data; thus, we can find the (estimated) optimal hyperparameters.

Remember: **DO NOT** use the test set for hyperparameter tuning.

(Q2) Use [`sklearn.model_selection.cross_validate`](#) to find the best k (neighbors) value in

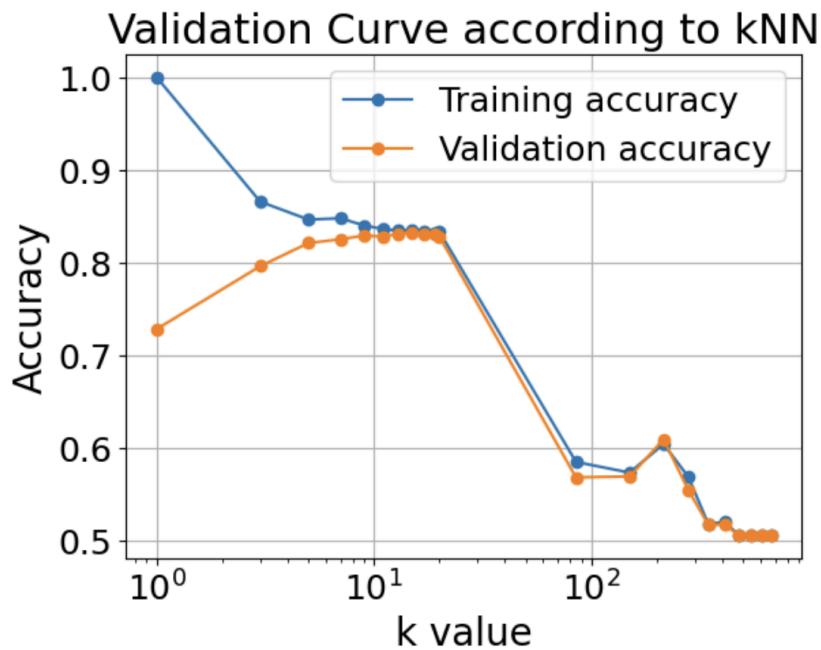
`list(range(1, 20, 2)) + list(range(20, 695, 65))` for predicting the `spread` class using `PCR_04` and `PCR_09`. Read the API carefully to understand how to extract train scores.

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.

Which k is the best? What are its average training and validation accuracies (estimated by cross-val.)? Which k values cause overfitting and underfitting and why?

Answer:



The best k value is the k that yields the highest validation accuracy together with the closest distance to the training accuracy. This behavior is obtained for $10 \leq k \leq 20$ and more the best one is at $k = 15$. The average training accuracy is 0.8347142857142857 and the average validation accuracy is 0.8320000000000001 .

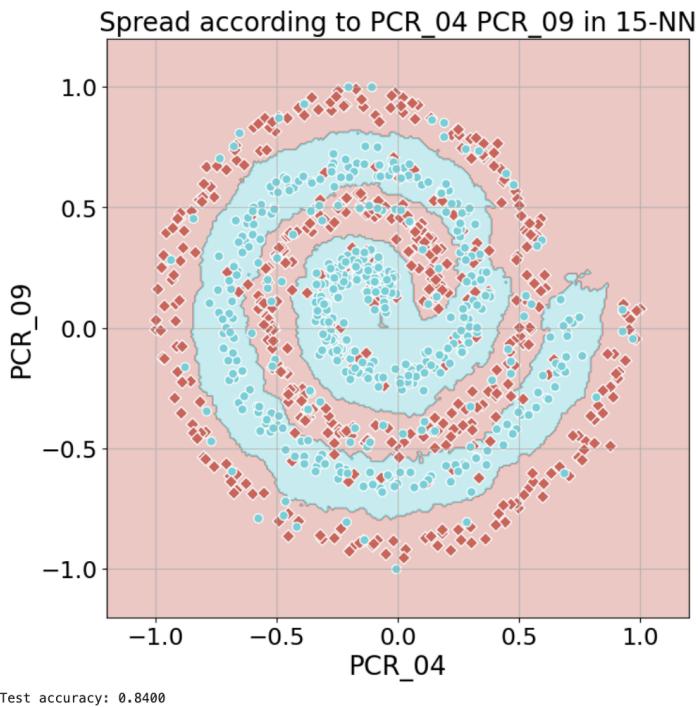
Overfitting is obtained at $k = 1$, because for this value the training data is learned

perfectly with accuracy 1, meaning the noise and outliers are considered in the learning although the validation accuracy is much lower meaning the model's generalization is flawed.

Underfitting is obtained at $k \geq 345$, because around these values, both the training and the validation accuracies are low. The high bias indicates we are not able to predict well neither the training data nor the validation (test) data.

- (Q3)** Use the optimal k value you found and retrain a k-NN model on all the training samples.

In your report: plot the decision regions of this final model (using `visualize_clf`) and write its test accuracy (computed on the separate test split) of this model.



- (Q4)** Compare the boundaries of the two models you have trained in (Q1) and (Q3).

Discuss the results and the exhibited behaviors (2-3 sentences).

Answer:

The boundaries are more continuous in the second model compared to the first.

This behavior is observed because the second model is less sensitive to outliers because the classification is according to the majority of 15 neighbors rather than one neighbor. It might make more mistakes on the training set but 15-NN is better for generalization. Test accuracy rose from 0.7560 in 1-NN to 0.84 in 15-NN.

Part 2: Decision trees

In this part we will be using `PCR_03` and `PCR_10` and `blood_type` (use it the same as in major hw1, split it into two groups: $\{O+, B+\}$ and $\{O-, A-, A+, B-, AB+, AB-\}$) to predict the `risk` class using decision trees. Rather than implementing the models by yourself, you will use `sklearn's` [DecisionTreeClassifier](#) with entropy as a splitting criterion (ID3) and focus on hyperparameter tuning and visualization.

Visualization

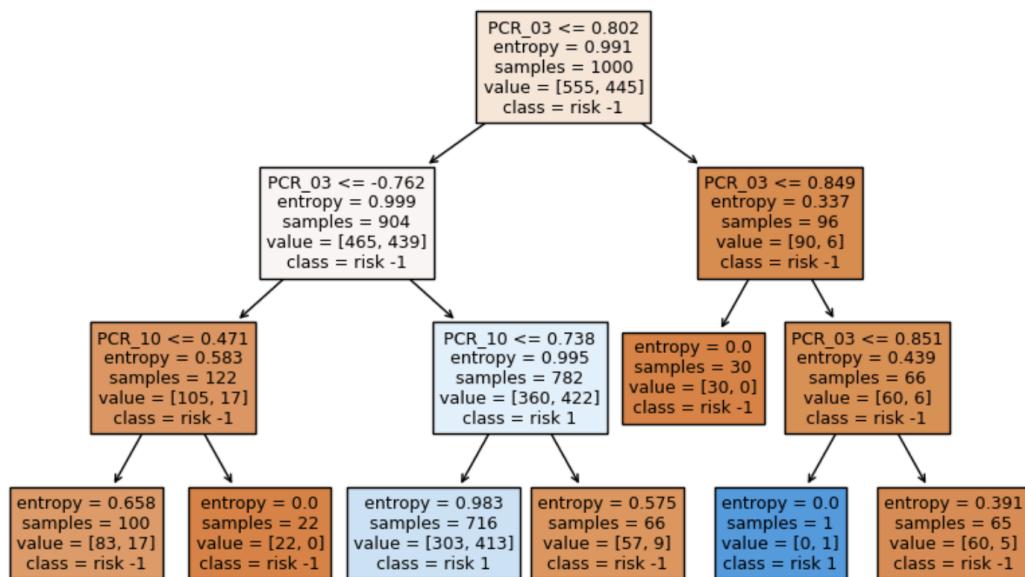
(Q5) Train a model with ID3 and `max_depth=3` (not including the root level; use the entire training set, i.e., all the features after preprocessing from all the training samples).

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!

Answer:

Train accuracy: 0.6660

Desicion Tree for risk based on PCR_03, PCR_10 and Blood Type



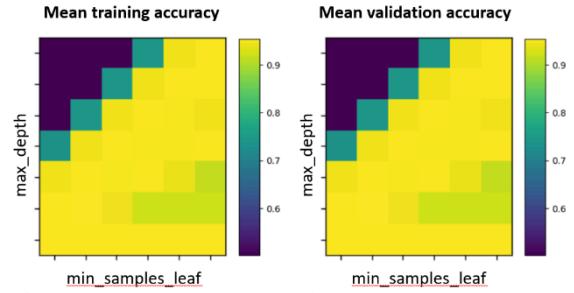
Model selection

It is time to search for the best tree to fight covid! Using the [DecisionTreeClassifier](#) documentation, understand how the `min_samples_leaf` argument can mitigate overfitting.

You will now tune two hyperparameters simultaneously – both `min_samples_leaf` and `max_depth`. You need to look for the combination of these two hyperparameters that lead to the best validation performance. There are many approaches for tuning multiple hyperparameters, and here we take the grid search approach (shortly explained [here](#)).

(Q6) Using 5-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:

- 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.
- Since we tune two hyperparameters, instead of a validation *curve*, plot two *heatmaps* ([seaborn](#) / [pyplot](#)), for each of the cross-validated training and validation accuracy (the heatmaps should roughly look like the ones to the right).



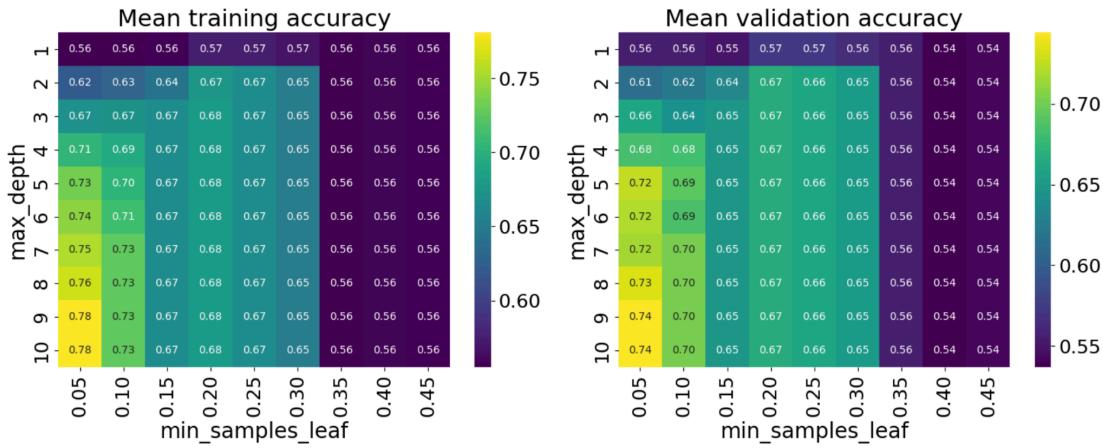
Plot the appropriate “ticks” on both axes and use annotations (`annot=True`) to explicitly write accuracies inside heatmap cells.

Important: The plots should be readable and informative!

- Add the 2 plots to your report and specify which hyperparameter combination is optimal.
- Write a hyperparameter-combination that causes underfitting.
- Write a hyperparameter-combination that causes overfitting.
- Add a short discussion regarding why each specific hyperparameter-combination from sub-questions 'd' and 'e' resulted in under/over-fitting.

ANSWER IS IN THE NEXT PAGE

Answer:



c.

The combination yielding the best validation error for predicting the `risk` class is
`{'max_depth': 4, 'min_samples_leaf': 0.20}`.

d. Hyperparameter-combination that causes underfitting is :

`{'max_depth': 2, 'min_samples_leaf': 0.45}`.

e. Hyperparameter-combination that causes overfitting is :

`{'max_depth': 10, 'min_samples_leaf': 0.05}`.

f.

Underfitting:

The mentioned combination creates a simple model which cannot explain the data properly. The classification is given after only two decisions or when the leaf size is 45% of the whole fold size. The classification might be more complex than this.

Overfitting:

The mentioned combination creates a complex model, which may be tuned according to outliers since we are giving class labels according to a small amount of samples and a deep tree. The class label is very specific and may be hard to generalize.

- (Q7) Write the number of hyperparameter combinations that were evaluated in your grid search. Had you wished to tune a third hyperparameter, how would that affect the number of combinations? Shortly discuss how searching over additional hyperparameters affects the total number of possible combinations.

Answer:

Number of hyperparameter combination: $A := |\text{minSampleLeaf}| \cdot |\text{maxDepth}| = 9 \cdot 10 = 90$

Since the order of the parameters is not significant, if we wished to tune a third hyperparameter we would have $|A| \cdot |\text{third param options}| = 90 \cdot |\text{third param options}|$.

All together, searching over additional hyperparameters is linear to each hyperparameter's options size.

- (Q8)** 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.

Answer:

For the optimal hyperparameter combination:

```
{'max_depth': 4, 'min_samples_leaf': 0.20}
```

The test accuracy is 0.6640.

Part 3: Linear SVM and the Polynomial kernel

In this part we will implement a Soft-SVM classifier to better understand this model. We will be predicting the `spread`, using only `PCR_04` and `PCR_09`. We will use gradient-based optimization to find the optimal parameter. Recall that using the whole dataset to perform one step update is costly. To mitigate this problem, we will implement the Stochastic Gradient Descent (SGD) algorithm.

Implementation of the loss and its gradient

Recall the Soft-SVM formulation:

$$\underset{\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}}{\operatorname{argmin}} \underbrace{\|\mathbf{w}\|_2^2 + C \sum_{i=1}^m \max\{0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\}}_{\approx p_C(\mathbf{w}, b)}$$

Following is the **analytic** sub-gradient of the objective function above

$$\nabla_{\mathbf{w}} p_C(\mathbf{w}, b) = 2\mathbf{w} + C \sum_{i=1}^m f(y_i(\mathbf{w}^\top \mathbf{x}_i + b))y_i \mathbf{x}_i, \text{ where } f(z) = \begin{cases} -1, & z < 1 \\ 0, & z \geq 1 \end{cases}$$
$$\frac{\partial}{\partial b} p_C(\mathbf{w}, b) = C \sum_{i=1}^m f(y_i(\mathbf{w}^\top \mathbf{x}_i + b))y_i$$

Task: Copy the `SoftSVM` module from the given `SoftSVM.py` into your notebook / project.

Task: Complete the (static) `SoftSVM.loss` method in the module, so that it computes the objective loss $p_C(w, b)$ on a given dataset.

Remember: `w` is a vector and `b` is a scalar.

Tip: When possible, prefer vector operations (e.g., `np.sum`, `np.sign`, `np.maximum`).

Avoid using for loops.

Task: Complete the (static) `SoftSVM.subgradient` method in the module, so that it computes the analytic sub-gradients $\nabla_w p_C(w, b)$ and $\frac{\partial}{\partial b} p_C(w, b)$ described above.

Tip: When possible, prefer vector operations (e.g., `np.sum`, `np.sign`, `np.maximum`).

Avoid using for loops.

Verifying your implementation: Numerical vs. analytical gradients

Recall from your calculus course, the definition of the derivative of a function $f: R \rightarrow R$ is:

$$f'(x_0) = \frac{f(x_0 + \delta) - f(x_0)}{\delta}$$

Thus, we can deduce a method to compute the **numerical** partial derivative w.r.t. w_i by approximating the limit expression with a finite δ :

$$\forall i = 1, \dots, d: \frac{\partial p_c}{\partial w_i} \approx \frac{p_c(w + \delta e_i b) - p_c(w, b)}{\delta} \triangleq u_i, \quad \text{where } e_i = [0, \dots, 0, 1_{\omega_{i-th}}, 0, \dots, 0]$$

Denote the **numerical** sub-gradient by $u_\delta(w, b) = [u_1 : u_d]$.

Using the numerical sub-gradient, we will now verify the correctness of your implementation for the loss and its analytic sub-gradient.

We will plot the residuals $\|\nabla_w p_c(w, b)_{\omega_{analytic}} - u_\delta(w, b)_{\omega_{numeric}}\|_2$ over many repeats as a function of δ .

Task: Copy the functions from the given `verify_gradients.py` into your notebook / project.

Read and understand these functions but do not edit them.

From this point, unless stated otherwise, we will use the features `PCR_04`, `PCR_09` to predict spread.

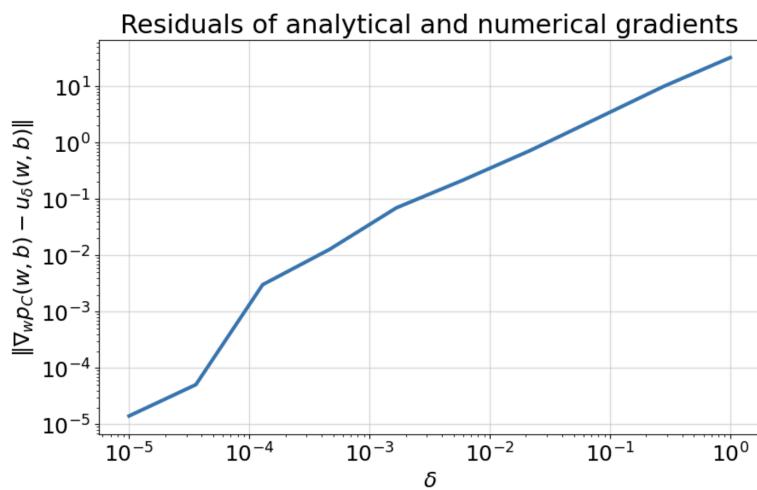
(Q9) Using `PCR_04`, `PCR_09`, generate a plot that compares the numerical gradients to the analytic gradients. Do this by running the following command:

```
compare_gradients(X_train, y_train, deltas=np.logspace(-5, 0, 10))
```

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

ANSWER IN THE NEXT PAGE

Answer:



The demonstrated behavior shows that for delta small enough,

$\nabla_w p_c(w, b)$ converges to $u_\delta(w, b)$ meaning each coordinate converges (by calculus3).

Solving Soft SVM problems using Stochastic Gradient Descent (SGD)

Task: Complete the given `SoftSVM.predict` method according to the decision rule of linear classifiers (return the predicted labels using the `{+1, -1}` convention).

Tip: prefer vector operations (e.g., `np.dot` and `np.sign`) when possible.

Avoid using `for loops`.

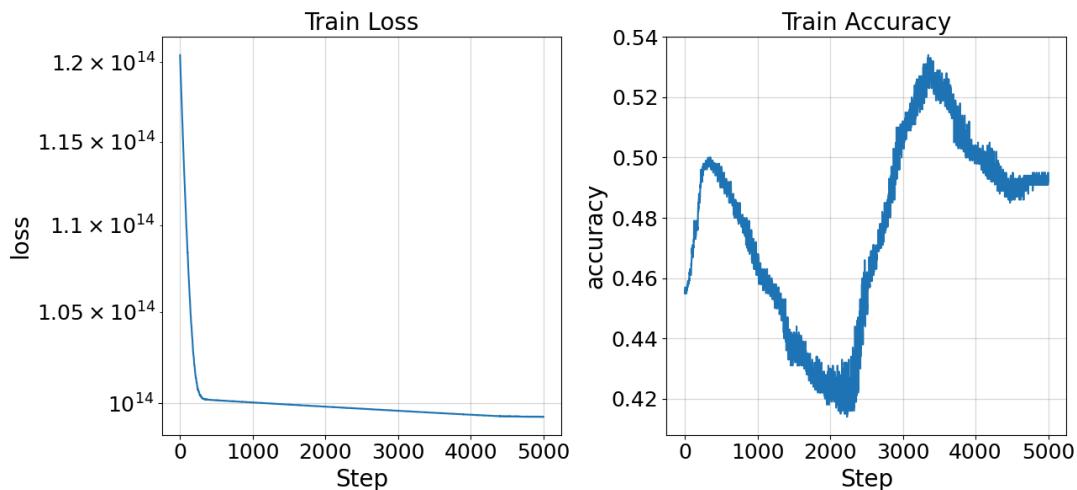
Task: Read and understand the given `SoftSVM.fit_with_logs` method.

Complete the code inside the loop to perform a gradient step (compute `g_w` and `g_b` and use them to update `w` and `b`).

Submit: Copy your completed `SoftSVM` module into a separate file called `SoftSVM.py`.

SGD is an iterative learning algorithm. A common method to analyze such algorithms is to plot a *learning curve*, i.e., plot the accuracy and/or loss of the model over time (i.e., steps).

(Q10) We trained a `SoftSVM` with `c = 1e11`, `lr = 2e-14` and obtained the following curves:

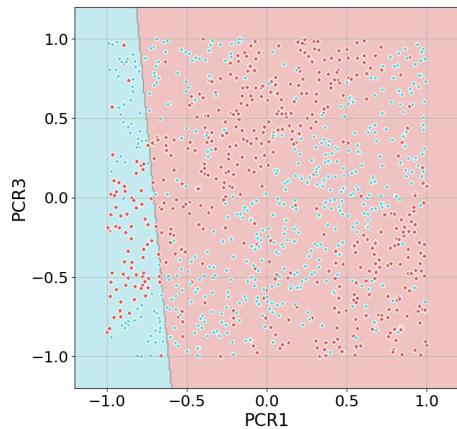


The resulting model induces the following decision regions (to the right).

Discuss the behavior of the loss and accuracy during the training of the model and any interactions you observe between the loss and the accuracy.

Does this interaction match your expectations?

If it does – explain why. If it does not – try to settle it.



Answer (in the next page):

ניתן לראות שפונקציות הלסן יורדת אֶל פונקציית הדיווק מבצעת תנודות. הינו מעריכים לראות שככל שהלוּס קטן הדיווק יגדל. ניתן להסביר תופעה זו ע"י כך שלקחו פרמטר C מאד גדול. פרמטר זה גורם לאלגוריתם לתת חשיבות רבה למרחק הנקודות מהמסוג. ככלומר הפרמטר גורם לכך שהאלגוריתם יעדיף לסווג לא נכון קבוצה גדולה של נקודות קרובות מאשר לסווג לא נכון קבוצה קטנה של נקודות רחוקות ובכך אנו יכולים לקבל דיוק פחות טוב למקרים שהלוּס קטן.

Using a feature mapping

We now want to understand the polynomial feature mapping and its corresponding kernel.

Recap: solving SVMs with feature mappings

To make our following explanation clearer, assume we want to use a 2nd-degree polynomial feature mapping on a 2-dimensional data (i.e., having two raw features).

We have two ways to use this feature mapping (we present Hard-SVM for simplicity):

1. Explicitly apply the feature mapping $\phi(x) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]$ and solve the primal problem in the new 6-dimensional space:

$$\underset{\mathbf{w} \in \mathbb{R}^6}{\operatorname{argmin}} \|\mathbf{w}\|_2^2 \text{ s.t. } y_i \cdot \mathbf{w}^\top \phi(\mathbf{x}_i) \geq 1, \forall i \in [m]$$

2. Use an appropriate kernel function, i.e., $K(u, v) = \phi(u)^\top \phi(v) = (u^\top v + 1)^2$, and solve the dual problem in an m -dimensional space (without explicitly computing the feature mappings):

$$\max_{\alpha \in \mathbb{R}_+^m} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \underbrace{\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)}_{=K(\mathbf{x}_i, \mathbf{x}_j)}$$

We will now use a **3rd degree** polynomial feature mapping to predict the spread using PCR_04 and 09. Since you only implemented the primal (Soft-)SVM problem, we will use the **first** way explained above and explicitly transform the 2 features using a 3rd degree polynomial mapping.

Task: We wish to understand the effect of the learning rate (step size) and make sure that our models converge.

The following snippet transforms the features, rescales them and then trains a SoftSVM model and plots the learning curves. Plot the a graph **for each learning rate** in the range `np.logspace(-11, -3, 5)` without changing the c value given below (if your graphs don't exhibit interesting phenomena, you may **slightly** modify the range, but make sure c is `1e5` across all learning rates; in such a case, explicitly mention in the following question that you changed the lr range).

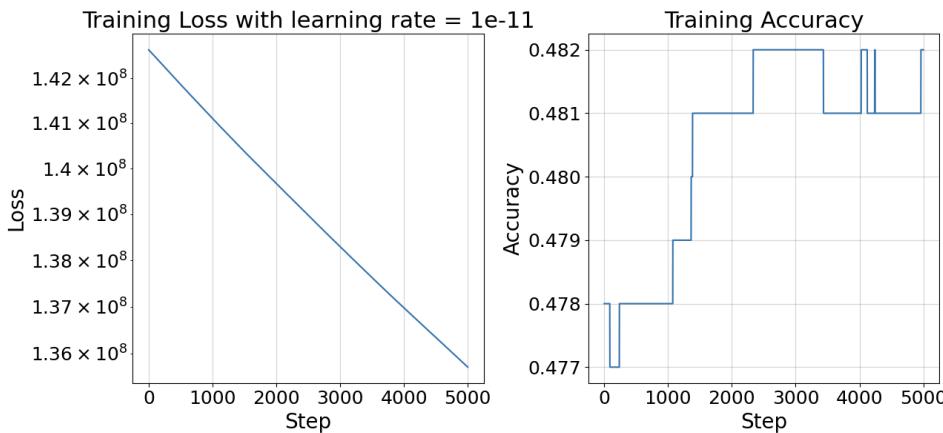
```
C=1e5
clf = SoftSVM(C=C, lr=lr)
X_train_poly = PolynomialFeatures(degree=3,).fit_transform(X_train)
X_train_poly = MinMaxScaler(feature_range=(-1,1)).fit_transform(X_train_poly)
losses, accuracies = clf.fit_with_logs(X_train_poly, y_train, max_iter=5000)
plt.figure(figsize=(13, 6))
plt.subplot(121), plt.grid(alpha=0.5), plt.title ("Training Loss")
plt.semilogy(losses), plt.xlabel("Step"), plt.ylabel("Loss")
plt.subplot(122), plt.grid(alpha=0.5), plt.title ("Training Accuracy")
plt.plot(accuracies), plt.xlabel("Step"), plt.ylabel("Accuracy")
plt.tight_layout()
plt.show()
```

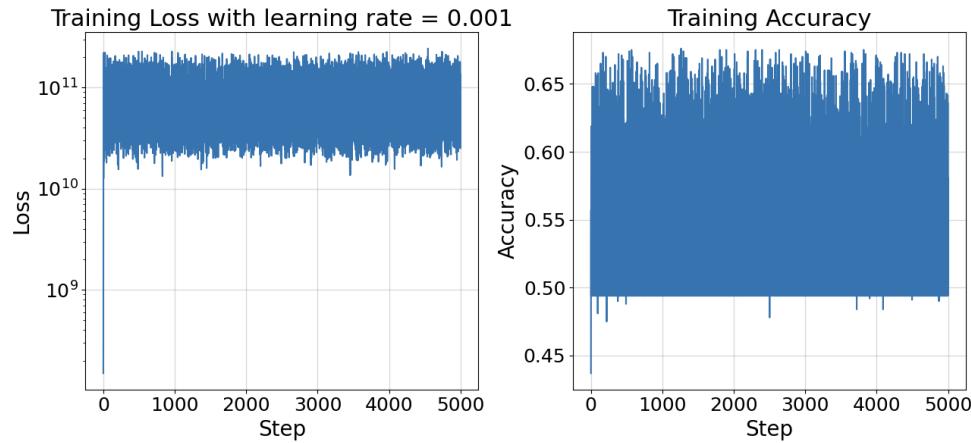
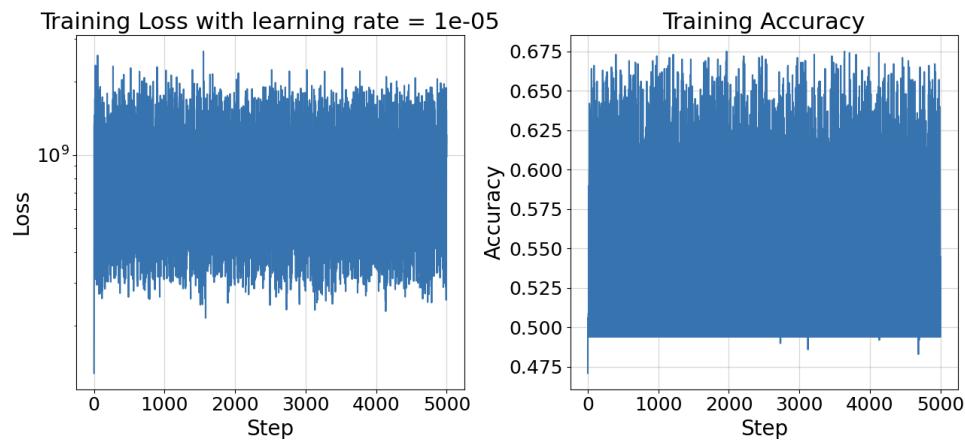
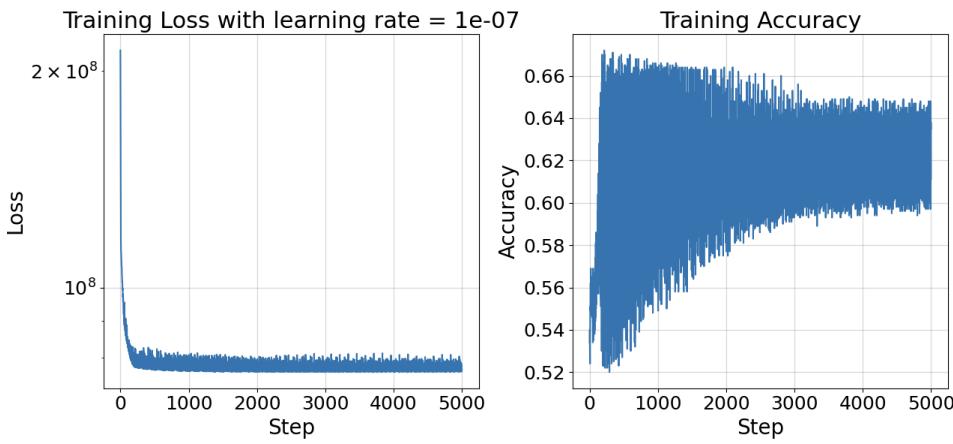
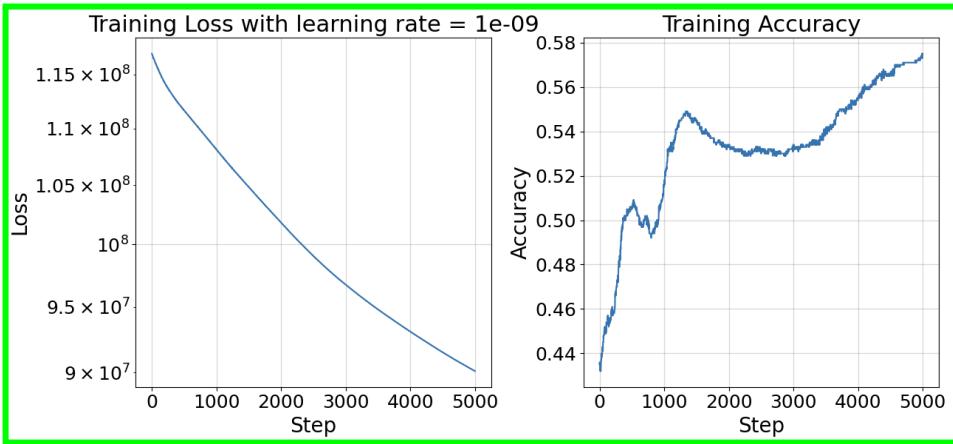
(Q11) Add the plots to the report and explain which learning rate you would choose and why.

Answer:

We would choose a learning rate `1e-09`. For this value, the loss is stably decreasing, slightly better than linear decay which is obtained for `lr=1e-11`. Along with that the training accuracy is increasing and reaching a good peak just a little after 1,000 steps. These two insights together produce a low bias which allows better generalization.

The plots for learning rates in `np.logspace(-11, -3, 5)`:





We now wish to create a single model that: (1) transforms the 2 features with a p -degree polynomial mapping; and then (2) normalizes the transformed data; and (3) trains your custom SoftSVM module on the data. To do so, we will use an [sklearn.pipeline.Pipeline](#).

The following code snippet creates such a pipeline and run fit:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
svm_clf = Pipeline([('feature_mapping', TODO),
                    ('scaler', MinMaxScaler()),
                    ('SVM', SoftSVM(C=..., lr=TODO))])
svm_clf.fit(X_train, y_train)
```

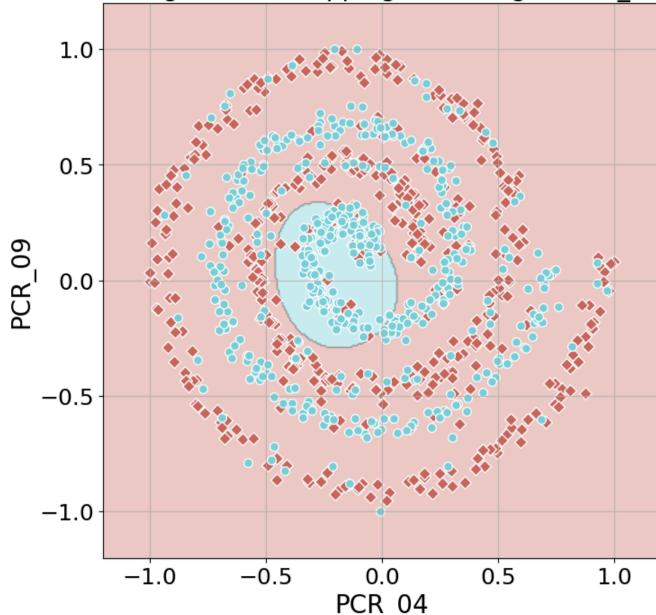
Task: Complete the pipeline above to make it apply a 3rd-degree [PolynomialFeatures](#) transformation. Use the learning rate you chose in (Q11). Train the model.

(Q12) In your report:

- Plot the trained model's decision regions.
- Write the respective training and test accuracies of the model.

Answer:

SoftSVM using Feature Mapping according to PCR_04 PCR_09



a.

b. Train accuracy: 0.6120 , Test accuracy: 0.6400

Part 4: The RBF kernel

Before we start: what is the Radial Basis Function kernel?

In Lectures 04 and 05 we presented the RBF kernel (also called the Gaussian kernel).

This kernel is often defined as $K(u, v) = \exp \exp \left\{ -\frac{1}{2\sigma^2} \|u - v\|_2^2 \right\}$ or $\exp \exp \left\{ -\gamma \|u - v\|_2^2 \right\}$, which can be decomposed using an infinite-dimensional feature mapping (see the lecture).

Note: If you wish, you can already practice this mapping by solving Q4 in Exam A from Winter 2021-22.

After solving the appropriate [2](#) optimization problem, we get a dual solution $\alpha \in \mathbb{R}_+^m$. Like we saw in the lecture, given a new datapoint $x \in \mathbb{R}^d$ for prediction, the model predicts

$$h(x) = \text{sign} \left(\sum_{i=1}^m \alpha_i y_i K(x, x_i) \right) = \text{sign} \left(\sum_{i \in [m], \alpha_i > 0} \alpha_i y_i K(x, x_i) \right).$$

This rule acts similarly to a weighted nearest-neighbor algorithm (on the training set), where “neighborhoods” of datapoints are computed using the kernel function.

For instance, consider a 1-dimensional training dataset: $\{(0, -1)_{\omega(x_1, y_1)}, (2, +1)_{\omega(x_2, y_2)}, (-1, +1)_{\omega(x_2, y_2)}\}$.

Assume the dual SVM solution is $\alpha = (1, 1, 2)$.

When $\sigma = 1$, we get the (weighted) Gaussians on Figure (a) below.

Given a new datapoint $x = 0.5$, the model predicts:

$$h(x) = \text{sign} \sum_i \alpha_i y_i K(x, x_i) = \text{sign}(1 \cdot (-1) \cdot K(0.5, 0) + 1 \cdot 1 \cdot K(0.5, 2) + 2 \cdot 1 \cdot K(0.5, -1)) = \text{sign}(-\exp \exp$$

In fact, we can create an even clearer visualization by drawing the weighted sum of positive points’ Gaussians (blue) and the one of negative ones (red) in Figure (b).

Finally, Figure (c) shows the case resulted Gaussians when $\sigma = \frac{1}{10}$. Notice how each training point influences only a small environment around it. We explain this below.

Figure (a)

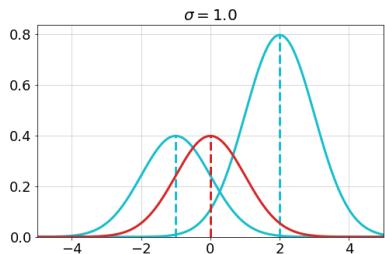


Figure (b)

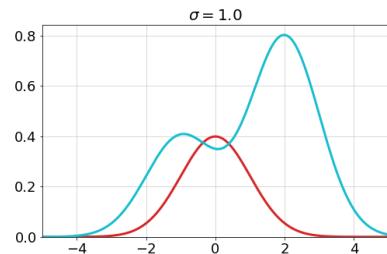
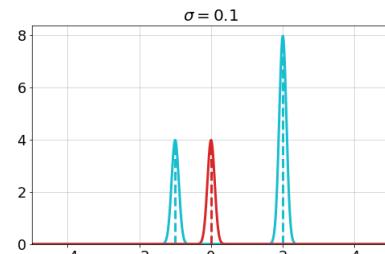


Figure (c)



Only very closely to $x = 0$,
the model predicts $y = -1$.

Each point affects only
a small environment around it

As another example, consider a (finite) solution vector $\alpha \in R_+^m$ in the extreme case

where $\sigma^2 \rightarrow 0$ and $\gamma \rightarrow \infty$, it can be easily shown that we should get a similar behavior (perhaps up to edge cases) to the 1-nearest-neighbor algorithm on the support vectors, i.e., the vectors with corresponding nonzero α weights:

$$h(x) = \text{sign} \left(\sum_{i \in [m], \alpha_i > 0} \alpha_i y_i K(x, x_i) \right) = \text{sign} \left(\sum_{i \in [m], \alpha_i > 0} \alpha_i y_i \exp \exp \left\{ -\gamma \|x - x_i\|_2^2 \right\} \right)$$

$$= \dots = y_i^*, \text{ where } i^* = \left\| x - x_i \right\|_2^2$$

To complete our explanations, read this [blogpost](#) and then watch this short [video](#).

While doing this, keep in mind the prediction rule $h(x) = \text{sign} \left(\sum_{i \in [m], \alpha_i > 0} \alpha_i y_i K(x, x_i) \right)$.

For the sake of simplicity, in the following question, we will introduce and use the Laplacian kernels:

$$K(u, v) = \exp \left\{ -\gamma \|u - v\|_2 \right\}$$

(Q13) Given a sample set $S = \{(x_i, y_i)\}_{i=1}^m$ where $x_i \in D$, $y_i \in \{-1, 1\}$, and $|\{x_i : y_i = 1\}| = \frac{m}{2}$:

- a. Dual Solution and Prediction Rule: Given the dual solution α such that $\forall i \in [m], \alpha_i = 1$, write the prediction rule $h(x)$ using the Laplacian kernels.

Answer a:

$$h(x) = \text{sign} \left(\sum_{i \in [m], \alpha_i > 0} \alpha_i y_i K(x, x_i) \right) = \text{sign} \left(\sum_{i \in [m]} y_i K(x, x_i) \right) = \text{sign} \left(\sum_{i \in [m]} y_i \exp \left\{ -\gamma \|x - x_i\|_2 \right\} \right).$$

when the first equality is by the definition of $h(x)$.

Now assume that there exist $\delta > 1$ that for every distinct $i, j \in [m]$, we have:

$$\|x_i - x_j\|_2 > 3\delta$$

Additionally, for every sample $x \in D$ there exists a tuple $(x_i, y_i) \in S$ such that:

$$y = y_i \text{ and } \|x - x_i\|_2 < \delta - 1$$

- b. Take the prediction rule from before and show the transition to the following form:

$$h(x) = \text{sign} \left(\sum_{j \in \{i \in [m] : y_i = 1\}} K(x, x_j) - \sum_{j \in \{i \in [m] : y_i = -1\}} K(x, x_j) \right)$$

Answer b:

From 13.a we have : $h(x) = \text{sign}\left(\sum_{i \in [m]} y_i K(x, x_i)\right)$

Hence:

$$\begin{aligned} h(x) &= \text{sign}\left(\sum_{i \in [m]} y_i K(x, x_i)\right) = \text{sign}\left(\sum_{j \in \{i \in [m] : y_i=1\}} y_i K(x, x_j) + \sum_{j \in \{i \in [m] : y_i=-1\}} y_i K(x, x_j)\right) = \\ &= \text{sign}\left(\sum_{j \in \{i \in [m] : y_i=1\}} 1 \cdot K(x, x_j) + \sum_{j \in \{i \in [m] : y_i=-1\}} (-1) \cdot K(x, x_j)\right) = \\ &= \text{sign}\left(\sum_{j \in \{i \in [m] : y_i=1\}} K(x, x_j) - \sum_{j \in \{i \in [m] : y_i=-1\}} K(x, x_j)\right) \end{aligned}$$

Given a new sample x with classification $y = 1$

c. Prove the following inequality: $\sum_{j \in \{i \in [m] : y_i=1\}} K(x, x_j) > \exp\{-\gamma(\delta - 1)\}$

Answer c:

Given that the classification of the sample x is $y=1$.

Also there exists a tuple $(x_k, y_k) \in S$ such that $y = y_k = 1$ and $\|x - x_k\|_2 < \delta - 1$.

Since the norm is non-negative we get:

$$\sum_{j \in \{i \in [m] : y_i=1\}} K(x, x_j) = (\sum_{k \neq j \in \{i \in [m] : y_i=1\}} K(x, x_j)) + K(x, x_k) > K(x, x_k) = \exp\left\{-\gamma\|x - x_k\|_2\right\} > \exp\{-\gamma(\delta - 1)\}$$

d. Prove Another Inequality: $\sum_{j \in \{i \in [m] : y_i=-1\}} K(x, x_j) \leq \frac{m}{2} \exp\{-\gamma(2\delta - 1)\}$

Hint: You can prove this using the norm triangle inequality or geometric arguments.

Answer d:

Given that the classification of the sample x is $y=1$.

Also there exists a tuple $(x_k, y_k) \in S$ such that $y = y_k = 1$ and $\|x - x_k\|_2 < \delta - 1$.

It is also known that for every distinct $i, j \in [m] : \|x_i - x_j\|_2 > 3\delta$.

All together :

$$3\delta < \|x_i - x_k\| \leq \|x - x_i\| + \|x - x_k\| < \|x - x_i\| + \delta - 1$$

Thus :

$$2\delta + 1 < \|x - x_i\|$$

Deducing to the formula:

$$\sum_{j \in \{i \in [m] : y_i = -1\}} K(x, x_j) = \frac{m}{2} K(x, x_j) = \frac{m}{2} \exp \left\{ -\gamma \|x - x_j\| \right\} \leq \frac{m}{2} \exp \left\{ -\gamma(2\delta - 1) \right\}$$

e. Show the Prediction Result: Given $\gamma = \ln \left(\frac{m}{2} \right)$, show that $h(x) = 1$.

Answer e:

$$\text{As we previously showed: } h(x) = \text{sign} \left(\sum_{j \in \{i \in [m] : y_i = 1\}} K(x, x_j) - \sum_{j \in \{i \in [m] : y_i = -1\}} K(x, x_j) \right).$$

From 13.c and 13.d :

$$\begin{aligned} & \sum_{j \in \{i \in [m] : y_i = 1\}} K(x, x_j) - \sum_{j \in \{i \in [m] : y_i = -1\}} K(x, x_j) > \exp \{-\gamma(\delta - 1)\} - \frac{m}{2} \exp \{-\gamma(2\delta - 1)\} = \\ &= \exp \left\{ -\frac{m}{2}(\delta - 1) \right\} - \frac{m}{2} \exp \left\{ -\frac{m}{2}(2\delta - 1) \right\} = e^{-\ln(\frac{m}{2})(\delta-1)} - \frac{m}{2} e^{-\ln(\frac{m}{2})(2\delta-1)} = \\ &= \frac{2}{m}^{(\delta-1)} - \frac{m}{2} \frac{2}{m}^{(2\delta-1)} = \frac{2}{m}^{(\delta-1)} \left(1 - \frac{2}{m}^{(\delta-1)} \right) > 0 \end{aligned}$$

$$\text{This means that } h(x) = \text{sign} \left(\sum_{j \in \{i \in [m] : y_i = 1\}} K(x, x_j) - \sum_{j \in \{i \in [m] : y_i = -1\}} K(x, x_j) \right) = 1$$

Notice that the last inequality is due to:

- $0 < \frac{2}{m} < 1$
- $\delta > 1 \Rightarrow \delta - 1 > 0$
- $0 < \frac{2}{m}^{(\delta-1)} < 1 \Rightarrow 0 < 1 - \frac{2}{m}^{(\delta-1)} < 1$

f. Prove the General Prediction Rule: Based on the previous proofs, show that the prediction rule $h(x)$ is always correct under the given assumptions.

Answer f:

We will show that if x is a sample with classification $y = -1$, then $h(x) = -1$.

Applying the same method we similarly have:

$$\sum_{j \in \{i \in [m] : y_i = -1\}} K(x, x_j) > \exp \{-\gamma(\delta - 1)\}, \quad \sum_{j \in \{i \in [m] : y_i = 1\}} K(x, x_j) \leq \frac{m}{2} \exp \{-\gamma(2\delta - 1)\}$$

Using the same arguments we have that :

$$\sum_{j \in \{i \in [m] : y_i = 1\}} K(x, x_j) - \sum_{j \in \{i \in [m] : y_i = -1\}} K(x, x_j) < 0, \text{ Hence:}$$

$$h(x) = \text{sign} \left(\sum_{j \in \{i \in [m] : y_i = 1\}} K(x, x_j) - \sum_{j \in \{i \in [m] : y_i = -1\}} K(x, x_j) \right) = -1$$

This gives us that the model is always correct under the given assumptions.

Finally, let us use a Kernel SVM with an RBF kernel to predict the spread using PCR_04,09. Since the corresponding feature mapping is of an infinite dimensionality, we will have to use the kernel trick and solve the dual problem, like we explained in (2). Our custom SoftSVM class is not suitable for solving dual formulations, hence we will use sklearn's implementation.

In the following questions, we recommend that you try to understand the exhibited behaviors in the plots considering the explanations at the beginning of this part.

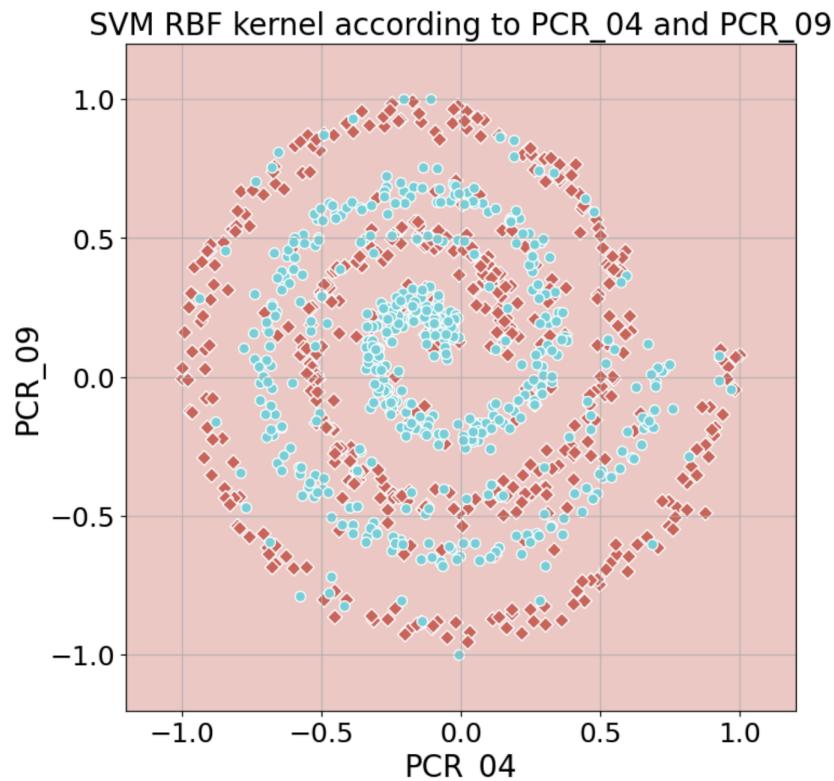
(Q14) Use sklearn.svm.SVC to train an SVM with an RBF kernel on the two features and the spread variable, with c=1 and $\gamma = 1e-7$. Plot the model's decision regions and include this plot in your report. Additionally, provide the training and test accuracies of the model. Based on these accuracies, would you classify the model as underfitting, overfitting, or as a reasonable fit?

Answer:

Train accuracy: 0.5060

Test accuracy: 0.4760

It is clear that the model is underfitting, it can be obtained by low train and test accuracies.



(Q15) Use `sklearn.svm.SVC` to train an SVM with an RBF kernel on the two features and the `spread` variable, with `c=1` and $\gamma = 200$. Plot the model's decision regions and include this plot in your report. Additionally, provide the training and test accuracies of the model. Compare the decision regions of this model with the k-NN model with k optimal from (Q3). We expected the two models to be very similar, but we should still see significant differences. Try to explain the differences (there isn't a single correct answer).

Hint: think what happens numerically when $\gamma = 200$ and $\|x - x_i\|_2^2$ is not very small.

Answer:

Current model accuracies:

Train accuracy: 0.8580

Test accuracy: 0.8240

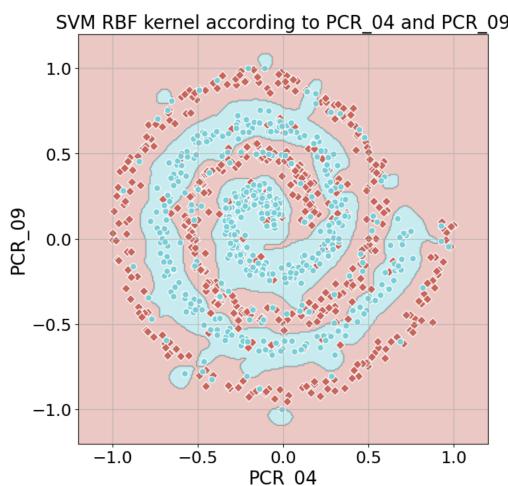
Optimal k (k=15) model accuracies:

Train accuracy: 0.8350

Test accuracy: 0.8400

It can be seen that there was an improvement in train accuracy for the SVM RBF kernel model although there is a slight worse performance in testing for the SVM model than the kNN model.

This might be explained by the values of gamma. As we seen in the lecture, higher gamma means that a single training example has a large influence, while a lower gamma means that the influence is more spread out. Thus, high value of gamma make the model very sensitive to outliers and to fit them to the trained separator. This can potentially overfit decision boundaries to classify perfectly the training data although the generalization may be damaged. The drop in test accuracy indicates a slight overfit of the model.



(Q16) Use [`sklearn.svm.SVC`](#) to train an SVM with an RBF kernel on the two features and the `spread` variable, with `C=1` and $\gamma = 5000$. Plot the model's decision regions and include this plot in your report. Additionally, provide the training and test accuracies of the model. Based on these accuracies, would you classify the model as underfitting, overfitting, or as a reasonable fit?

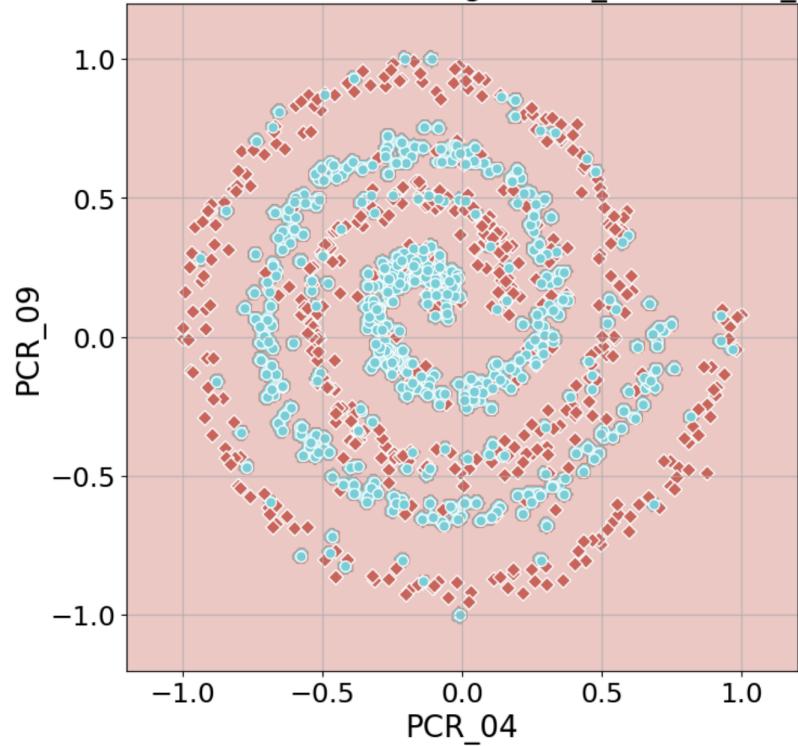
Answer:

Train accuracy: 0.9800

Test accuracy: 0.6920

Since the train accuracy is almost perfect but the test accuracy is far from perfect, we can infer the model is overfitting and lacks generalization.

SVM RBF kernel according to PCR_04 and PCR_09



Submitting the files

Return to the instructions on Pages 2-3 and make sure you submit all required files.