

## HW2: Algorithm Implementation and Basic Model Selection

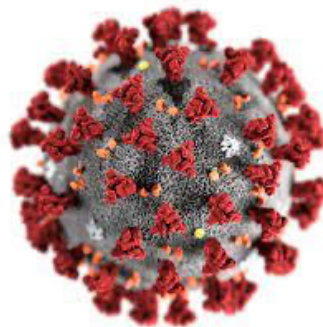
### Goal

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, 9.1.2025, 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**

- **Students in reserve service (מילואים) who have received approval via email from the main Teaching Assistant (TA) may skip questions 7, 8, 10, 12, 13 section 2 and 15. At the beginning of their final report, these students must include a screenshot of the email approval.**
- 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 file (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

For example, in HW1, we start with the simple k-NN model and use it to practice new concepts, such as model selection. Here we use k-NN on `PCR_01` and `PCR_08s` 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_01` and `PCR_08`.

**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 including accuracy and labels) for train and test sets.

Notice: From now on, when asked to use `visualize_clf`, plot the decision regions with the train and test partitions or visualize two plots for each partition.

### 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_01` and `PCR_08`. 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  $k$  is the best? What are its average training and validation accuracies (estimated by cross-val.)? Which  $k$  values cause overfitting and underfitting and why?

- (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).

## Part 2: Decision trees

In this part we will be using `PCR_04` and `PCR_06` 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!

## 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.

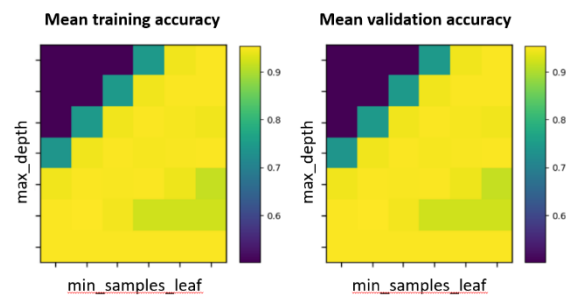
**Important:** The plots should be readable, informative, and at least 15 by 15!

- 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.

Students in reserve service (מילואים) with approval from the main TA can skip questions 7 and 8.

**(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.

**(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.



## 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_01` and `PCR_08`. 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:

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

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

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

**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  $p_C(\mathbf{w}, b)$  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  $\nabla_{\mathbf{w}} p_C(\mathbf{w}, b)$  and  $\frac{\partial}{\partial b} p_C(\mathbf{w}, b)$  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  $f: \mathbb{R} \rightarrow \mathbb{R}$  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  $\delta$ :

$$\forall i = 1, \dots, d: \frac{\partial p_C}{\partial w_i} \approx \frac{p_C(\mathbf{w} + \delta \mathbf{e}_i, b) - p_C(\mathbf{w}, b)}{\delta} \triangleq u_i, \text{ where } \mathbf{e}_i = [0, \dots, 0, \underbrace{1}_{i\text{-th}}, 0, \dots, 0]$$

Denote the **numerical** sub-gradient by  $\mathbf{u}_\delta(\mathbf{w}, b) = \begin{bmatrix} u_1 \\ \vdots \\ u_d \end{bmatrix}$ .

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  $\| \underbrace{\nabla_{\mathbf{w}} p_C(\mathbf{w}, b)}_{\text{analytic}} - \underbrace{\mathbf{u}_\delta(\mathbf{w}, b)}_{\text{numeric}} \|_2$  over many repeats as a function of  $\delta$ .

**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_01`, `PCR_08` to predict spread.

**(Q9)** Using `PCR_01`, `PCR_08`, 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.

## 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.

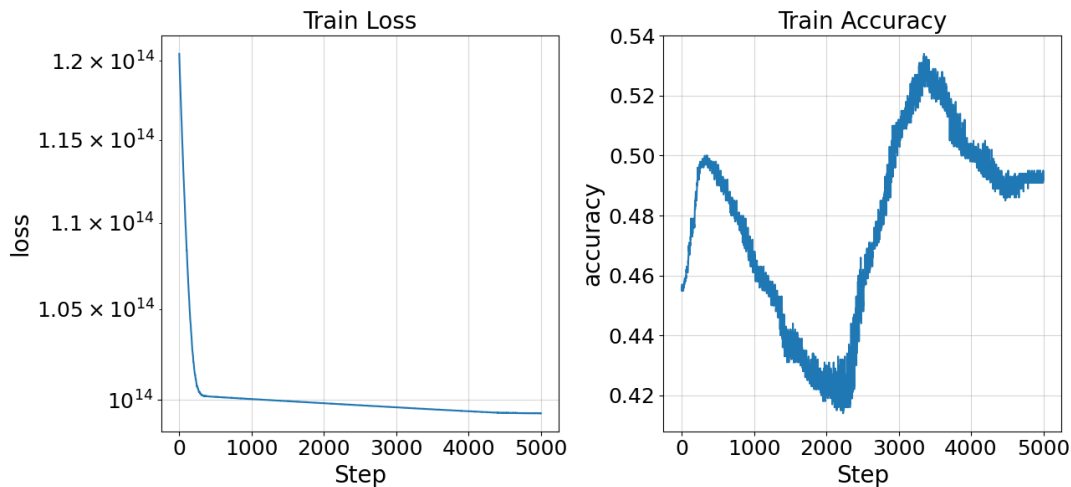
Complete the code inside the loop to perform a gradient step (compute  $g_w$  and  $g_b$  and use them to update  $w$  and  $b$ ).

**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).

Students in reserve service (מילואים) with approval from the main TA can skip question 10.

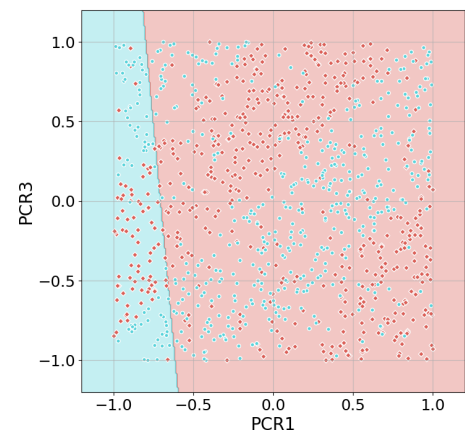
**(Q10)** We trained a `SoftSVM` with  $c = 1e11$ ,  $lr = 2e-14$  and obtained the following curves:



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.

**Important:** no more than 4 sentences, you are supposed to focus on the main objective of learning.



## 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:

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

2. Use an appropriate kernel function, i.e.,  $K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u})^\top \phi(\mathbf{v}) = (\mathbf{u}^\top \mathbf{v} + 1)^2$ , 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 **3<sup>rd</sup> degree** polynomial feature mapping to predict the spread using PCR\_01 and 08. 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 3<sup>rd</sup> 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 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.

**Students in reserve service (מילואים) with approval from the main TA can skip question 12.**

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=1e5, lr=TODO))])
svm_clf.fit(X_train, y_train)
```

**Task:** Complete the pipeline above to make it apply a 3<sup>rd</sup>-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.

## 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  $K(\mathbf{u}, \mathbf{v}) = \exp\left\{-\frac{1}{2\sigma^2} \|\mathbf{u} - \mathbf{v}\|_2^2\right\}$  or  $\exp\{-\gamma \|\mathbf{u} - \mathbf{v}\|_2^2\}$ , 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  $\alpha \in \mathbb{R}_+^m$ . As we saw in the lecture, given a new datapoint  $\mathbf{x} \in \mathbb{R}^d$  for prediction, the model predicts  $h(\mathbf{x}) = \text{sign}(\sum_{i=1}^m \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)) = \text{sign}(\sum_{i \in [m], \alpha_i > 0} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i))$ .

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:  $\left\{ \underbrace{(0, -1)}_{(x_1, y_1)}, \underbrace{(2, +1)}_{(x_2, y_2)}, \underbrace{(-1, +1)}_{(x_2, y_2)} \right\}$ .

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

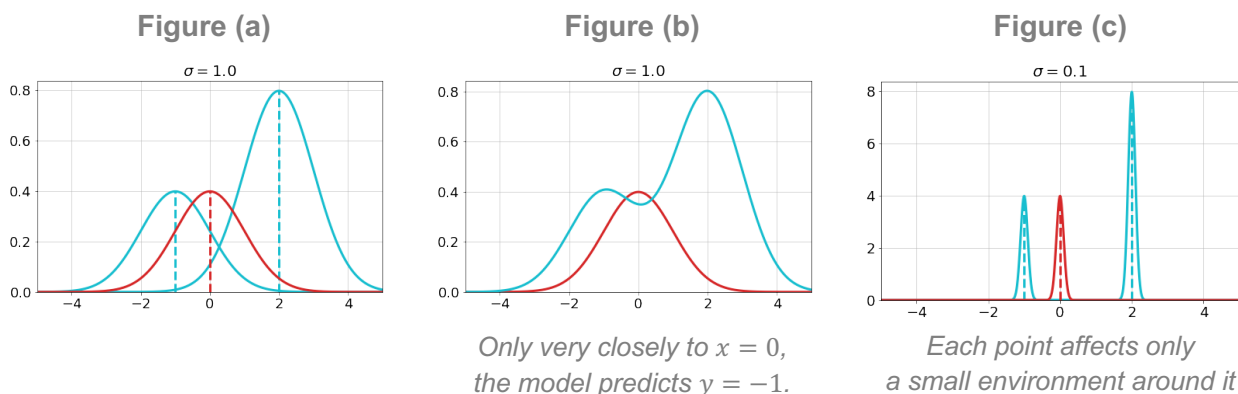
When  $\sigma = 1$ , we get the (weighted) Gaussians on Figure (a) below.

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

$$\begin{aligned} h(\mathbf{x}) &= \text{sign} \sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) = \text{sign}(1 \cdot (-1) \cdot K(0.5, 0) + 1 \cdot 1 \cdot K(0.5, 2) + 2 \cdot 1 \cdot K(0.5, -1)) \\ &= \text{sign} \left( -\exp\left\{-\frac{\|0.5 - 0\|_2^2}{2}\right\} + \exp\left\{-\frac{\|0.5 - 2\|_2^2}{2}\right\} + 2 \exp\left\{-\frac{\|0.5 - (-1)\|_2^2}{2}\right\} \right) \\ &= \text{sign} \left( -\exp\left\{-\frac{0.25}{2}\right\} + \exp\left\{-\frac{2.25}{2}\right\} + 2 \exp\left\{-\frac{2.25}{2}\right\} \right) \approx \text{sign}(-0.88 + 0.32 + 0.65) = +1 \end{aligned}$$

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.



As another example, consider a (finite) solution vector  $\alpha \in \mathbb{R}_+^m$  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  $\alpha$  weights:

$$h(\mathbf{x}) = \text{sign} \left( \sum_{i \in [m], \alpha_i > 0} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) \right) = \lim_{\gamma \rightarrow \infty} \text{sign} \left( \sum_{i \in [m], \alpha_i > 0} \alpha_i y_i \exp\{-\gamma \|\mathbf{x} - \mathbf{x}_i\|_2^2\} \right) \\ = \dots = y_{i^*}, \text{ where } i^* = \underset{i \in [m], \alpha_i > 0}{\text{argmin}} \|\mathbf{x} - \mathbf{x}_i\|_2^2$$

To complete our explanations, read this [blogpost](#) and then watch this short [video](#). While doing this, keep in mind the prediction rule  $h(\mathbf{x}) = \text{sign}(\sum_{i \in [m], \alpha_i > 0} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i))$ .

**(Q13)** Kernels provide a powerful method for working with complex objects such as strings, graphs, and more. Kernels built by recursively combining one or more of the following rules are well-defined kernels:

1.  $k(x, x') = e^{k_1(x, x')}$
2.  $k(x, x') = f(x)k_1(x, x')f(x')$

Where  $k_1$  is a well-defined kernel for example,  $k(x, x') = x^T x'$ , and  $f$  is any function.

### **Section 1:**

We have a dataset of emails, and for each email  $i$ , we represent it as a **set**  $S_i \in \Omega$  containing all the **unique** words in the email, for example, the email:

"The first rule of Fight Club is: you do not talk about Fight Club"

It is represented as:

$$S = \{ \text{"the", "first", "rule", "of", "fight", "club", "is", "you", "do", "not", "talk", "about"} \}.$$

The word "Club" appears twice in the mail, but the representative set has one element; that's how sets work.

We aim to develop a classifier that classifies each email as "spam" or "not spam" using a kernel method. We'll use the following kernel function to measure the similarity between two emails  $S_i$  and  $S_j$ :

$$K_{\text{spam}}(S_i, S_j) = e^{|S_i \cap S_j|}$$

We wish to define a proper feature mapping (here is a good place to [start](#)), given we know that the number of all the unique words in email space  $\Omega$  is  $d$  we represent words by index  $i \in [d]$ , we define the following feature mapping  $\phi: \Omega \rightarrow \mathbb{R}^d$  as:

$$(\phi(S))_i = \begin{cases} 1, & i \in S \\ 0, & o.w \end{cases}$$

- Using the vector dot product show that  $\langle \phi(S_i), \phi(S_j) \rangle = |S_i \cap S_j|$  and explain why it's a well-defined kernel.
- Now prove that  $K_{spam}$  is a well-defined kernel.

Students in reserve service (מילואים) with approval from the main TA can skip question 13 section 2.

### **Section 2:**

Your friend told you it's not sophisticated enough and suggested the following kernel:

$$K_{spam \text{ pro max}} = e^{|S_i \cap S_j| - |S_i \cup S_j|}$$

- First, let's refresh our memory on Set theory. Given two finite groups  $S_i, S_j$  the size of the union is  $|S_i \cup S_j| = |S_i| + |S_j| - |S_i \cap S_j|$ .  
Define the function  $f(S_i) = e^{-\|\phi(S_i)\|_1}$  and prove that  $e^{-|S_i \cup S_j|}$  is a kernel.
- Prove that if  $k_1(x, x')$  and  $k_2(x, x')$  are kernels then  $k(x, x') = k_1(x, x') k_2(x, x')$  is a kernel.
- Prove that  $K_{spam \text{ pro max}}$  is a well-defined kernel.
- Give a general explanation (no more than 3 sentences) of why the suggested kernel is a better choice.
- Can you suggest an even better kernel? (It isn't supposed to be well-defined).

Finally, let us use a Kernel SVM with an RBF kernel to predict the spread using PCR\_01,08. Since the corresponding feature mapping is of an infinite dimensionality, we will have to use the kernel trick and solve the dual problem, as 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?

Students in reserve service (מילואים) with approval from the main TA can skip question 15.

**(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.

**(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?

## Submitting the files

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