HW2: Algorithm Implementation and Basic Model Selection

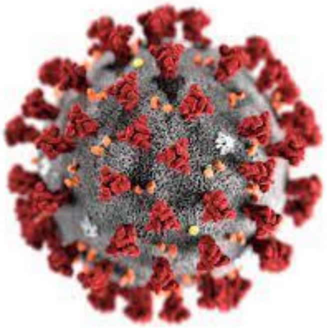
Goal

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, 14.03.2024, 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* or *id1-id2-id3.*
    - 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).
            - *prepare.py* : your up-to-date data preparation pipeline (including normalization).
        - Also:
            - Working with jupyter: your notebook, *<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).

   Note: We do not mean that you should restore features that should have been transformed into other features and only then deleted, like the `symptoms` feature.

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_01` and `PCR_03` 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_03`.

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, 871, 85))` for predicting the `spread` class using `PCR_01` and `03`. 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 focus on predicting 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

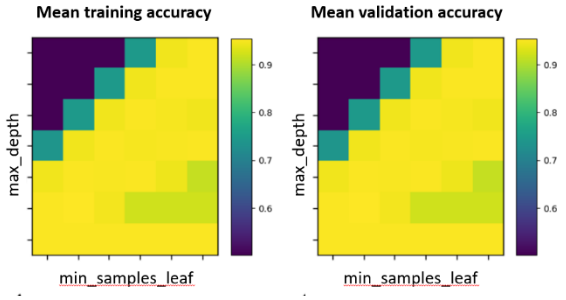
It is time to search for the best tree to fight covid! Using the DecisionTreeClassifier documentation, understand how the `min_samples_leaf` argument can mitigate overfitting.

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:

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.

b. 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!

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

d. Write a hyperparameter-combination that causes underfitting.

e. Write a hyperparameter-combination that causes overfitting.

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

(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_03`. 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:

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

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

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

$$\frac{\partial}{\partial b} p_C(\boldsymbol{w}, b) = C\sum_{i=1}^{m} f\left(y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\right) y_i$$

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 $f: \mathbb{R} \to \mathbb{R}$ 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 δ:

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

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

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 $\| \underbrace{\nabla_{\boldsymbol{w}} p_C(\boldsymbol{w}, b)}_{\text{analytic}} - \underbrace{\boldsymbol{u}_\delta(\boldsymbol{w}, b)}_{\text{numeric}} \|_2$ 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_01, PCR_03 to predict spread.

(Q9) Using PCR_01, PCR_03, 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, -1, 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.

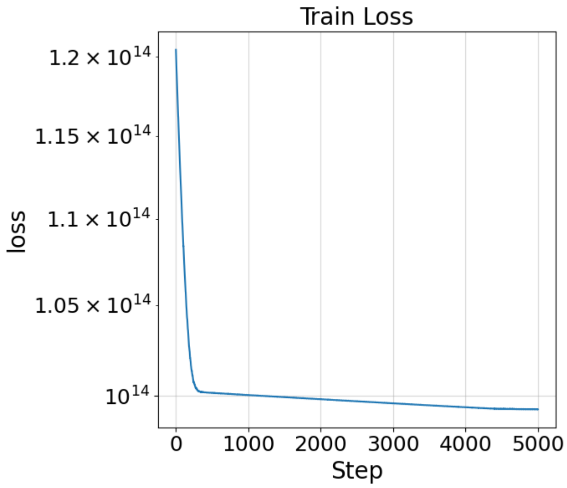
Task: Read and understand the given `SoftSVM.fit_with_logs` method.

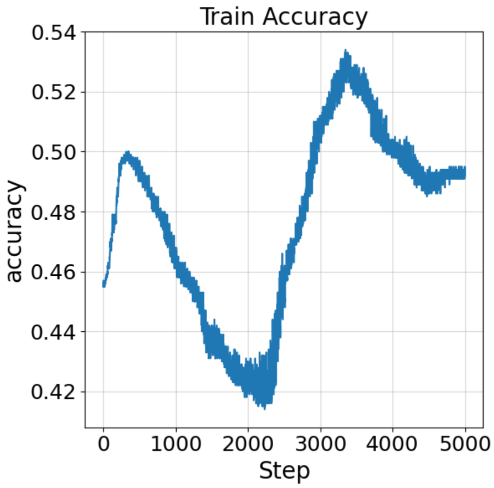
Complete the code inside the loop to perform a gradient step (compute `g_w` and `g_b` and use them to update `w` and `b`).

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).

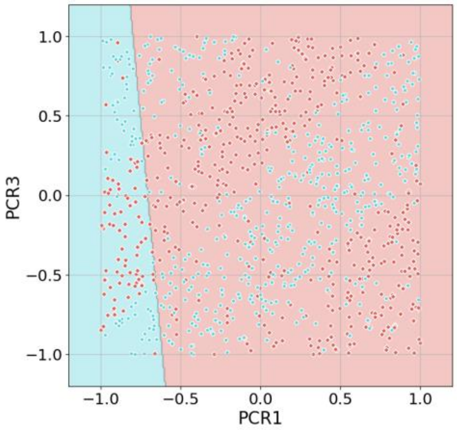
(Q10) We trained a `SoftSVM` with `C = 1e11`, `lr = 2e-14` and obtained the following curves:





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.



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* $\phi(x) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2\right]$ *and solve the primal problem in the new 6-dimensional space:*

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

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

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

We will now use a 3rd **degree** polynomial feature mapping to predict the `spread` using `PCR_01` and `03`. 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(-9, -5, 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)
Xtrain_poly = PolynomialFeatures(degree=3,).fit_transform(Xtrain)
Xtrain_poly = MinMaxScaler(feature_range=(-1,1)).fit_transform(Xtrain_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.

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:

```
from sklearn.preprocessing import MinMaxScaler
svm_clf = Pipeline([('feature_mapping', TODO),
                    ('scaler', MinMaxScaler()),
                   ('SVM', SoftSVM(C=1e5, lr=TODO))])
```

We wish to set the SVM's `max_iter` to 5,000. Since you work with a pipeline, use the following way to specify an argument of an inner model inside the pipeline:

```
svm_clf.fit(X_train, y_train, SVM__max_iter=5000)
```

where "SVM" is the name of the final step of your pipeline.

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:

a. Plot the trained model's decision regions.
b. 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 $K(\boldsymbol{u},\boldsymbol{v}) = \exp\left\{-\frac{1}{2\sigma^2}\|\boldsymbol{u}-\boldsymbol{v}\|_2^2\right\}$ *or* $\exp\{-\gamma\|\boldsymbol{u}-\boldsymbol{v}\|_2^2\}$*, 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 $\boldsymbol{\alpha} \in \mathbb{R}_+^m$*. Like we saw in the lecture, given a new datapoint* $\boldsymbol{x} \in \mathbb{R}^d$ *for prediction, the model predicts* $h(\boldsymbol{x}) = \operatorname{sign}(\sum_{i=1}^m \alpha_i y_i K(\boldsymbol{x},\boldsymbol{x}_i)) = \operatorname{sign}\left(\sum_{i\in[m],\alpha_i>0} \alpha_i y_i K(\boldsymbol{x},\boldsymbol{x}_i)\right)$.

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

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

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

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

$$h(\boldsymbol{x}) = \operatorname{sign}\sum_i \alpha_i y_i K(\boldsymbol{x},\boldsymbol{x}_i) = \operatorname{sign}(1\cdot(-1)\cdot K(0.5,0) + 1\cdot 1\cdot K(0.5,2) + 2\cdot 1\cdot K(0.5,-1))$$

$$= \operatorname{sign}\left(-\exp\left\{-\frac{\|0.5-0\|_2^2}{2}\right\} + \exp\left\{-\frac{\|0.5-2\|_2^2}{2}\right\} + 2\exp\left\{-\frac{\|0.5-(-1)\|_2^2}{2}\right\}\right)$$

$$= \operatorname{sign}\left(-\exp\left\{-\frac{0.25}{2}\right\} + \exp\left\{-\frac{2.25}{2}\right\} + 2\exp\left\{-\frac{2.25}{2}\right\}\right) \approx \operatorname{sign}(-0.88+0.32+0.65) = +1$$

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)

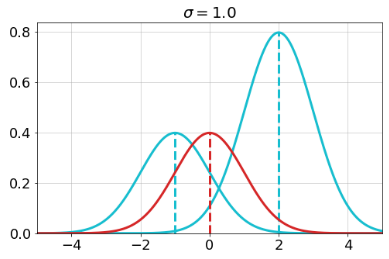


Figure (b)

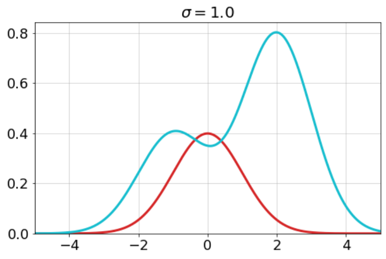
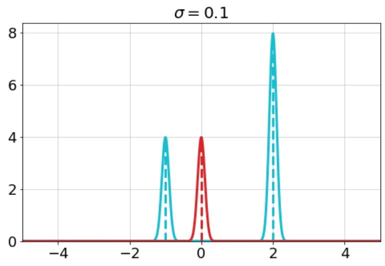


Figure (c)



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 $\boldsymbol{\alpha} \in \mathbb{R}_+^m$ in the extreme case where $\sigma^2 \to 0$ and $\gamma \to \infty$, 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:

$$h(\boldsymbol{x}) = \operatorname{sign}\left(\sum\nolimits_{i\in[m],\alpha_i>0} \alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i)\right) = \lim_{\gamma\to\infty} \operatorname{sign}\left(\sum\nolimits_{i\in[m],\alpha_i>0} \alpha_i y_i \exp\{-\gamma\|\boldsymbol{x} - \boldsymbol{x}_i\|_2^2\}\right)$$

$$= \cdots = y_{i^*}, \text{ where } i^* = \operatorname{argmin}_{i\in[m],\alpha_i>0}\|\boldsymbol{x} - \boldsymbol{x}_i\|_2^2$$

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

(Q13) We want to explore the other extreme, where $\sigma^2 \to \infty$ and $\gamma \to 0$.

a. Prove the following statement:

$$\lim_{\gamma\to 0} \operatorname{sign}\left(\sum\nolimits_{i\in[m],\alpha_i>0} \alpha_i y_i \exp\{-\gamma\|\boldsymbol{x} - \boldsymbol{x}_i\|_2^2\}\right) = \operatorname{argmax}_{y\in\{-1,+1\}} \sum\nolimits_{\{i|y_i=y\}} \alpha_i$$

Assume:

i. In our case $\lim(\text{sign}) = \text{sign}(\lim)$.

Note that this assumption does not hold in the general case for $\lim(\text{sign})$.

ii. The norms of all feature vectors $\boldsymbol{x}$ are bounded, i.e., $\exists c_1 < \infty$ such that $\|\boldsymbol{x}\|_2 \le c_1$.

iii. The norm of $\boldsymbol{\alpha}$ is bounded as well, i.e., $\exists c_2 < \infty$ such that $\|\boldsymbol{\alpha}\|_2 \le c_2$.

b. Now assume $\alpha_i = 1, \forall i \in [m]$.

Based on the statement you have proved, explain the decision rule of an SVM model with an RBF kernel when $\gamma \to 0$. Specify a model (that we've learned about) other than SVM that exhibits a similar (but not identical) behavior.

Finally, let us use a Kernel SVM with an RBF kernel to predict the `spread` using `PCR_01,03`. 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 in light of the explanations at the beginning of this part. You are not required to explain these behaviors in your report.

(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 γ =`1e-7`. Plot the model's decision regions (attach to your report). Does the decision boundary match the decision rule discussed in (Q13)b?

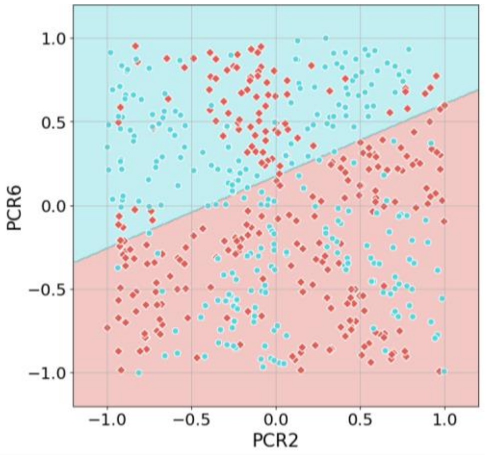
(Q15) Use sklearn.svm.SVC to train an SVM with an RBF kernel on the two features and the `spread` variable, with `C=1` and $\gamma = 5000$. Plot the model's decision regions (attach to your report). Compare the decision regions of this model with the k-NN model with $k = 1$ from (Q1). We expected the 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 $\gamma = 5000$ and $\|\boldsymbol{x} - \boldsymbol{x}_i\|_2^2$ is not very small.

Part 5: Custom Feature Mapping

In this part we will implement a custom feature mapping to predict the `risk` label, using the features `PCR_02` and `PCR_06`.

(Q16) Create temporary dataframes (for both train and test) only containing patients whose blood type is **not** "`O+`" or "`B+`" (recall the `SpecialProperty` feature), having the features `PCR_02` and `PCR_06`, and the `risk` label (only). Attach an `sns.jointplot` of the 2 features (on the train set) to your report, and make sure the data points are colored by label. Write a brief description of the visual pattern you observe in the data. Note: if you have trouble recognizing a pattern, try a different marker (dot) size, i.e., change the `s` parameter of the `jointplot`.

Given: We trained a linear SVM model with `C=1` using only features `PCR_02,06` to predict the `risk` label and obtained the decision regions attached on the right. The training and test accuracies are 58% and 53% respectively.



Task: Noticing the pattern of the 2D datapoints, we hypothesize that mapping the datapoints to the angle between the feature vector and one of the axes might help improve accuracy. Transform the features `PCR_02` and `PCR_06` to their angle from one of the axes and store the result in a column, either in the same dataframe or in a separate one (don't forget to include the labels).

Notes:

a. The angle should be in radians.
b. Hint: `numpy` trigonometric functionality can help. Be sure to read the documentation carefully to understand the return values of the functions you use.

(Q17) The following snippet visualizes the distribution of the new mapped feature. It uses sns.scatterplot and sns.kdeplot together in the same plot. Run the code (you may need to adjust it to your variable and column names) and attach the plot to your report. Make sure the plot is readable. Shortly describe the plot. Does the new angle feature seem to be linearly separable after the mapping?

```
plt.figure(figsize=(15, 5))
sns.scatterplot(data=Xtrain, x='angle', y=[0]*Xtrain.shape[0], hue='risk', s=20)
sns.kdeplot(data= Xtrain, x='angle', hue='risk', common_norm=True, fill=False)
plt.ylim(bottom=-1e-2)
```

(Q18) Use the following snippet to create another column in your dataframe, that calculates the sine of the angle to which you mapped the original features:

```
beta=1
Xtrain['sin_angle'] = np.sin(beta*Xtrain['angle'])
```

After creating the new feature, plot an `sns.jointplot` of the two mapped features (the angle and its sine). Do these two features together seem linearly separable?

(Q19) Propose a way to adjust the `beta` variable in the snippet above so that the 2D feature mapping (the combination of the angle and its sine) becomes more easily separable by a linear model.

Apply your idea and train another `LinearSVC` model only on the updated mapping to predict the `risk` label (don't forget to normalize the mapped features before training the linear model). In your report, write the training and test accuracies of the model. Plot the decision regions and attach the plot to your report. How does this model perform compared to the given linear model (in the previous page)?

Submitting the files

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