Laboratory 9

In this laboratory we will focus on Support Vector Machines for binary classification tasks.

Linear SVM

Support Vector Machines are linear classifiers that look for maximum margin separation hyperplanes. The (primal) SVM objective consists in minimizing

$$J(\boldsymbol{w}, b) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\max(0, 1 - z_i(\boldsymbol{w}^T\boldsymbol{x}_i + b))$$

where n is the number of training samples, C is a hyper-parameter and z_i is the class label for the i-th sample, encoded as

$$z_i = \begin{cases} +1 & \text{if } x_i \text{ belongs to class } \mathcal{H}_T \\ -1 & \text{if } x_i \text{ belongs to class } \mathcal{H}_F \end{cases}$$

As we have seen, to solve the SVM problem we can also consider the dual formulation

$$J^D(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{H}\boldsymbol{\alpha} + \boldsymbol{\alpha}^T\mathbf{1}$$

$$\text{s. t. } 0 \leq \boldsymbol{\alpha}_i \leq C\ , \forall i \in \{1 \dots n\}\ , \quad \sum_{i=1}^{n}\boldsymbol{\alpha}_i z_i = 0$$

where $\mathbf{1}$ is a n-dimensional vector of ones, and $\boldsymbol{H}$ is a matrix whose elements are

$$\boldsymbol{H}_{ij} = z_i z_j \boldsymbol{x}_i^T\boldsymbol{x}_j$$

The SVM dual solution is the maximizer of $J^D(\boldsymbol{\alpha})$. The dual and primal solutions $\boldsymbol{\alpha}^*$ and $\boldsymbol{w}^*$ are related through

$$\boldsymbol{w}^* = \sum_{i=1}^{n}\boldsymbol{\alpha}_i^* z_i \boldsymbol{x}_i$$

and the optimal bias b^* can be computed considering a sample $\boldsymbol{x}_i$ that lies on the margin:

$$z_i(\boldsymbol{w}^{*T}\boldsymbol{x}_i + b^*) = 1$$

and then solving for b^*. As we have discussed, the dual problem also allows for non-linear separation through the kernel trick. Indeed, we can replace dot-products $\boldsymbol{x}_i^T\boldsymbol{x}_j$ with kernel functions $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, so that $\boldsymbol{H}_{ij} = z_i z_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and we can compute $\boldsymbol{w}^{*T}\boldsymbol{x}$ through kernel functions:

$$\boldsymbol{w}^{*T}\boldsymbol{x} = \sum_{i=1}^{n}\boldsymbol{\alpha}_i^* z_i k(\boldsymbol{x}_i, \boldsymbol{x}) = \sum_{i|\boldsymbol{\alpha}_i \neq 0}\boldsymbol{\alpha}_i^* z_i k(\boldsymbol{x}_i, \boldsymbol{x})$$

The latter expressions can be used, for example, when solving for b^* and when we want to evaluate the score of a test sample $\boldsymbol{x}$.

For linear SVMs, we can optimize either the primal or dual formulation. Unfortunately, the unconstrained formulation of the primal objective function is non-differentiable. While the L-BFGS method may still be able to find the minimizer of J, we have *no guarantee* that the algorithm will stop close to the optimal value of $(\boldsymbol{w}, b)$. Ad-hoc numerical solvers have been developed, however they are out of the scope of our course.

The dual formulation is differentiable, however it contains both box constraints (i.e. constraints of the form $a \leq \alpha_i \leq b$) and the additional constraint $\sum_{i=1}^{n}\alpha_i z_i = 0$. The L-BFGS algorithm that we employed is able to handle box constraints, however it cannot incorporate the latter.

The constraint $\sum_{i=1}^{n}\alpha_i z_i = 0$ derives from the presence of the bias term in the SVM primal formulation. We therefore slightly modify the SVM problem as to make the constraint disappear. In this way

we will be able to employ L-BFGS-B (the B stands for box-constraints) to solve the dual problem.

We reformulate the primal problem as the minimization of

$$\widehat{J}(\widehat{\boldsymbol{w}}) = \frac{1}{2}\|\widehat{\boldsymbol{w}}\|^2 + C\sum_{i=1}^{n}\max(0, 1 - z_i(\widehat{\boldsymbol{w}}^T\widehat{\boldsymbol{x}}_i))$$

where

$$\widehat{\boldsymbol{x}}_i = \begin{bmatrix}\boldsymbol{x}_i \\ 1\end{bmatrix}, \quad \widehat{\boldsymbol{w}} = \begin{bmatrix}\boldsymbol{w} \\ b\end{bmatrix}$$

We can observe that $\widehat{\boldsymbol{w}}^T\widehat{\boldsymbol{x}}_i = \boldsymbol{w}^T\boldsymbol{x} + b$, i.e. the scoring rules have the same expression as the original formulation. However, in contrast with the original SVM problem, we are also regularizing the value of the bias term, since we regularize the norm of $\widehat{\boldsymbol{w}}$:

$$\|\widehat{\boldsymbol{w}}\|^2 = \|\boldsymbol{w}\|^2 + b^2$$

Regularization of the bias can, in general, lead to sub-optimal decisions in terms of separating margin. We can mitigate this effect by using a mapping

$$\widehat{\boldsymbol{x}}_i = \begin{bmatrix}\boldsymbol{x}_i \\ K\end{bmatrix}$$

As K becomes larger, the effects of regularizing b become weaker. However, as K becomes larger, the dual problem also becomes harder to solve (i.e. the algorithm may requires many additional iterations).

The dual objective of the modified primal SVM becomes the maximization of

$$\widehat{J}^D(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}^T\widehat{\boldsymbol{H}}\boldsymbol{\alpha} + \boldsymbol{\alpha}^T\mathbf{1}$$
$$\text{s. t. } 0 \leq \boldsymbol{\alpha}_i \leq C \ , \forall i \in \{1 \dots n\}$$

i.e., the same formulation as before but without the equality constraint and with matrix $\widehat{\boldsymbol{H}}$ computed from the extended features $\widehat{\boldsymbol{x}}_i$ rather than from the original features $\boldsymbol{x}_i$:

$$\widehat{\boldsymbol{H}}_{i,j} = z_i z_j \widehat{\boldsymbol{x}}_i^T \widehat{\boldsymbol{x}}_j$$

Write a function that computes the primal SVM solution through the dual SVM formulation $\widehat{J}^D$.

Suggestions:

- You can work directly with vectors $\widehat{\boldsymbol{x}}$. Build the extended matrix of training data that contains all training samples. For K = 1, you would compute

$$\widehat{\boldsymbol{D}} = \begin{bmatrix}\boldsymbol{x}_1 & \boldsymbol{x}_2 & \dots & \boldsymbol{x}_n \\ 1 & 1 & \dots & 1\end{bmatrix}$$

- Compute $\widehat{\boldsymbol{H}}$. You can use `for` loops. However, you can speed up computations exploiting `numpy.dot` to compute the matrix $\widehat{\boldsymbol{G}}$ all products $\widehat{\boldsymbol{G}}_{ij} = \widehat{\boldsymbol{x}}_i^T\widehat{\boldsymbol{x}}_j$ from $\widehat{\boldsymbol{D}}$ in a single call, and broadcasting to compute $\widehat{\boldsymbol{H}}$ from $\widehat{\boldsymbol{G}}$.

- The `scipy` implementation of L-BFGS-B computes the minimizer of a function, but we want to maximize $\widehat{J}^D(\boldsymbol{\alpha})$. We can cast the problem as **minimization** of

$$\widehat{L}^D(\boldsymbol{\alpha}) = -\widehat{J}^D(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\widehat{\boldsymbol{H}}\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\mathbf{1}$$
$$\text{s. t. } 0 \leq \boldsymbol{\alpha}_i \leq C \ , \forall i \in \{1 \dots n\}$$

- Write the function that, given a numpy array `alpha` containing values for vector $\boldsymbol{\alpha}$, computes $\widehat{L}^D(\boldsymbol{\alpha})$, and its gradient (although the gradient may be automatically computed, consider that $\boldsymbol{\alpha}$ has a number of elements equal to the number of samples, and the number of additional function evaluations would therefore be significantly higher). The gradient of $\widehat{L}^D(\boldsymbol{\alpha})$ is

$$\nabla_{\boldsymbol{\alpha}}\widehat{L}^D = \widehat{\boldsymbol{H}}\boldsymbol{\alpha} - \mathbf{1}$$

  Remember to re-shape the gradient so that it's a 1-D `numpy` array of shape `(n,)`

- Use `scipy.optimize.fmin_l_bfgs_b` to minimize $\widehat{L}^D$. You have to specify the box constraints $0 \leq \boldsymbol{\alpha}_i \leq C$. These can be provided through argument `bounds` of `scipy` function `scipy.optimize.fmin_l_bfgs_b` . `bounds` should be a list of pairs `(min, max)`. Each element of the list corresponds to a different optimization variable. In our case, the list should have N elements, and should be

```
[(0, C), (0, C), ...  (0, C)]
```

- Once you have computed the dual solution, you can recover the primal solution through

$$\widehat{\boldsymbol{w}}^* = \sum_{i=1}^{n} \boldsymbol{\alpha}_i z_i \widehat{\boldsymbol{x}}_i$$

α → PRIMAL SOLUTION
W → DUAL SOLUTION

- To classify a pattern, you have to compute the score $\widehat{\boldsymbol{w}}^{*T}\widehat{\boldsymbol{x}}_t$. You can either compute the extended data matrix for the evaluation set, or extract the terms $\boldsymbol{w}^*, b^*$ from $\widehat{\boldsymbol{w}}^*$ and then compute the solution as $\boldsymbol{w}^{*T}\boldsymbol{x}_t + b^*$ (notice that, if you choose the latter option and set $K \neq 1$, then you need to appropriately re-scale the value of b^* to ensure that $\widehat{\boldsymbol{w}}^{*T}\widehat{\boldsymbol{x}}_t = \boldsymbol{w}^{*T}\boldsymbol{x}_t + b^*$)

SCORE

- The SVM decision rule is to assign a pattern to class $\mathcal{H}_T$ if the score is greater than 0, and to $\mathcal{H}_F$ otherwise. However, SVM decisions are not probabilistic, and are not able to account for different class priors and mis-classification costs. Bayes decisions thus require either a score post-processing step, i.e. score calibration, or cross-validation to select the optimal threshold for a specific application. Below we simply use threshold 0 and compute the corresponding accuracy.

- Hyper-parameter C should be selected through cross-validation. Below you can find results for different values of C.

- You can check the dual solution is correct by computing the duality gap. At the optimal solution, we have $\widehat{J}(\widehat{\boldsymbol{w}}^*) = \widehat{J}^D(\boldsymbol{\alpha}^*)$. Implement a function that computes the primal objective

$$\widehat{J}(\widehat{\boldsymbol{w}}^*) = \frac{1}{2}\left\|\widehat{\boldsymbol{w}}^*\right\|^2 + C\sum_{i=1}^{n}\max(0, 1 - z_i(\widehat{\boldsymbol{w}}^{*T}\widehat{\boldsymbol{x}}_i))$$

$J^D(\alpha) = -\hat{J}^D(\alpha)$

$\hat{J}(\hat{w}^*) = -J^D(\alpha)$

and compute the duality gap

$$\widehat{J}(\widehat{\boldsymbol{w}}^*) - \widehat{J}^D(\boldsymbol{\alpha}^*) = \widehat{J}(\widehat{\boldsymbol{w}}^*) + \widehat{L}^D(\boldsymbol{\alpha}^*)$$

$-J^D(\alpha) + J^P(\alpha)$

The smaller the duality gap, the more precise is the dual (and thus the primal) solution.

- You can control the precision of the L-BFGS solution through the parameter `factr` . The default value is `factr=10000000.0` . Lower values result in more precise solutions (i.e. closer to the optimal solution), but require more iterations. **Below `factr=1.0` was used.**

- The `scipy` implementation of L-BFGS-B calls the objective function a maximum of 15000 times, and the algorithm stops when this threshold is reached. You can specify a larger amount for the maximum number of calls through the `maxfun` argument

- You can also control the maximum number of allowed iterations through the argument `maxiter`

USA LA SOGLIA A ZERO

The following table reports primal and dual objectives for the solution returned by L-BFGS with different values of C and K on IRIS, with `factr=1.0`. The accuracy on the evaluation set is also reported for cross-checking (the predictions are obtained comparing the SVM score with a threshold $t = 0$). The training and evaluation data are obtained using the same splits of previous laboratories . Higher values of K correspond to weaker regularization of the SVM bias term.

NOTE: Due to numerical precision and the fact that L-BFGS-B is not well suited for the problem, you may obtain slightly different values for the dual and primal loss, and, depending on these values, significantly different values for the duality gap. In any case, if the duality gap is small (few orders of magnitude lower than the objective itself), there's good chance that the implementation is correct. In rare cases, you may also obtain very small, negative duality gaps. Also in this case this is due to

I VALORI DIPENDONO DALLA MACCHINA

```
self.alpha, self.primal, _ = scipy.optimize.fmin_l_bfgs_b(func=self.__LDc_obj,
                                                          bounds=self.bounds,
                                                          x0=np.zeros(Dtrain.shape[1]),
                                                          factr=1.0)
```

SET DI VARIABILI CHE MINIMIZZA LA FUNZIONE OBIETTIVO

VALORE MINIMO DELLA FUNZIONE OBIETTIVO CIOÈ IL VALORE DELLA FUNZIONE OBIETTIVO CALCOLATO USANDO IL SET DI VAR CHE LO MINIMIZZA

NOI STIAMO CALCOLANDO IL VALORE DELLA FUNZIONE DUALE CHE È PIÙ SEMPLICE DA RISOLVERE E MENO COSTOSO.

- MATRICE H

IN SVM H È LA MATRICE DI KERNEL CHE VIENE USATA NEL CALCOLO DELL'OBIETTIVO DI MINIMIZZAZIONE
SI PUÒ CALCOLARE IN 2 MODI

1) NEL CASO DI KERNEL POLINOMIALE O RBF, H È CALCOLATA COME IL PRODOTTO TRA KER E SELF.LTRAIN_2_MATRIX. DOVE KER È MATRICE QUADRATA IN CUI L'ELEMENTO KER[i][j] È IL VALORE DEL KERNEL CALCOLATA TRA i-ESIMA E j-ESIMA OSSERVAZIONE DI ADDESTRAMENTO

- Polynomial kernel of degree d: $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\boldsymbol{x}_1^T \boldsymbol{x}_2 + c)^d$
- Radial Basis Function kernel: $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = e^{-\gamma \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2}$

X => DTR

2) SE NON USI IL KERNEL, H VIENE CALCOLATO COME PRODOTTO ELEMENTO PER ELEMENTO TRA G E LTRAIN2_MATRIX

VIENE SEMPRE UTILIZZATA PER IL CALCOLO DELL'OBIETTIVO

SELF.LTRAIN_2 : CONTIENE LE ETICHETTE DI CLASSE DI ADDESTRAMENTO TRASFORMATO IN MODO CHE I VALORI SIANO -1 O 1

SELF.LTRAIN_2_MATRIX: CONTIENE IL PRODOTTO DELLE ETICHETTE QUINDI L'ELEMENTO IN (i,j) È IL PRODOTTO DELLE ETICHETTE DEI CAMPIONI i,j

numerical error, and the duality gap should be close to the machine precision.

The table also reports minDCF and actual DCF. Since the SVM scores have no probabilistic interpretation, they may typically show significant miscalibration, especially when the application prior is far from the empirical training set prior. We can still compute minimum DCFs, and compute actual costs assuming that scores were LLRs, however we should keep in mind that using Bayes decisions for SVM scores may lead to very poor decisions and poor actual costs - on the other hand, we may also be lucky and obtain good decisions without any form of post-processing of the scores - a validation set can help us to in judging if score calibration is indeed required or not.

K	C	Primal loss	Dual loss	Duality gap	Error rate	minDCF ($\pi_T = 0.5$)	actDCF ($\pi_T = 0.5$)
1	0.1	3.774974e+00	3.774974e+00	6.641808e-07	2.9%	0.0556	0.0625
1	1.0	1.577994e+01	1.577993e+01	6.454718e-06	5.9%	0.0556	0.1181
1	10.0	7.896893e+01	7.896893e+01	1.888172e-06	5.9%	0.0556	0.1181
10	0.1	2.983576e+00	2.983576e+00	1.812922e-08	11.8%	0.0000	0.2222†
10	1.0	1.131707e+01	1.131707e+01	1.939987e-06	5.9%	0.0556	0.1181
10	10.0	5.464492e+01	5.464483e+01	8.859310e-05	5.9%	0.0625	0.1181

†This is potentially the best model (with an optimal threshold we would get a minDCF of 0, thus we could also achieve an error rate of 0%), but, due to poor threshold selection, we actually achieve a DCF of 0.2222, and an error rate of 11.8%, i.e. the worst results in the table

Kernel SVM

SVMs allow for non-linear classification through an implicit expansion of the features in a higher-dimensional space. The SVM dual objective depends on the training samples only through dot-products, and we can compute a classification score through scalar products between training and evaluation samples. Therefore, SVM does not require that we explicitly compute the feature expansion: it's sufficient that we are able to compute the scalar product between the expanded features $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{\phi}(\boldsymbol{x}_1)^T \boldsymbol{\phi}(\boldsymbol{x}_2)$. Function k is called *kernel function.*

Implement the dual SVM (without bias) classifier for generic kernel functions. You can re-use the previously developed code, however you should replace the computation of $\widehat{\boldsymbol{H}}$ with

$$\widehat{\boldsymbol{H}}_{i,j} = z_i z_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

In contrast with linear SVM, we are not able to compute the primal solution and its cost. However, we can compute the score of a test sample as

$$s(\boldsymbol{x}_t) = \sum_{i=1}^{n} \boldsymbol{\alpha}_i^* z_i k(\boldsymbol{x}_i, \boldsymbol{x}_t)$$

where the summation is taken over the *training* samples (in practice we can consider just the samples for which $\boldsymbol{\alpha}_i > 0$, i.e. the support vectors).

You can try different kernels:

- Polynomial kernel of degree d: $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\boldsymbol{x}_1^T \boldsymbol{x}_2 + c)^d$
- Radial Basis Function kernel: $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = e^{-\gamma \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2}$

The choice of the kernel and of its hyper-parameters (e.g. c and γ) can also be made through cross-validation.

NOTE: To add a (regularized) bias in the non-linear SVM version it's not sufficient to simply extend the feature vectors as we previously did, but we should rather add a constant value to our kernel function:

$$\widehat{k}(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(\boldsymbol{x}_1, \boldsymbol{x}_2) + \xi$$

For the linear SVM, we were doing this implicitly by expanding the feature vectors $\boldsymbol{x}_i$, which corresponded to $\xi = K^2$. For the dual problem, we work with the original features, and we add the constant term directly to the kernel evaluations.

Below you can find the dual objective value and the accuracy for different configurations.

SERVE A CONTROLLARE L'AMPIEZZA DEL KERNEL. UN VALORE ALTO AUMENTA L'AMPIEZZA DEL KERNEL RENDENDO IL MODELLO PIÙ SENSIBILE

$K = \sqrt{\xi}$	C	Kernel	Dual loss	Error rate	minDCF ($\pi_T = 0.5$)	actDCF ($\pi_T = 0.5$)
0.0	1.0	Poly ($d = 2, c = 0$)	6.663628e+00	8.8%	0.0625	0.1736
1.0	1.0	Poly ($d = 2, c = 0$)	6.296912e+00	8.8%	0.0625	0.1736
0.0	1.0	Poly ($d = 2, c = 1$)	3.592918e+00	2.9%	0.0556	0.0556
1.0	1.0	Poly ($d = 2, c = 1$)	3.569987e+00	2.9%	0.0556	0.0556
0.0	1.0	RBF ($\gamma = 1.0$)	1.198930e+01	8.8%	0.0625	0.1736
1.0	1.0	RBF ($\gamma = 1.0$)	1.197813e+01	8.8%	0.0625	0.1736
0.0	1.0	RBF ($\gamma = 10.0$)	1.842757e+01	11.8%	0.1736	0.2292
1.0	1.0	RBF ($\gamma = 10.0$)	1.839182e+01	8.8%	0.1736	0.1736

Project

NOTE: training SVM models may require some time. To speed-up the process, we suggest that you first setup the experiments for this project using only a fraction of the model training data. Once the code is ready, you can then re-run all the experiments with the full dataset.

BASSI VALORI DI C IMPLICANO UNA FORTE REGOLARIZZ. MENTRE ALTI AL CONTRARIO

1

Apply the SVM to the project data. Start with the linear model (to avoid excessive training time we consider only the models trained with K = 1.0). Train the model with different values of C. As for logistic regression, you should employ a logarithmic scale for the values of C. Reasonable values are given by `numpy.logspace(-5, 0, 11)`. Plot the minDCF and actDCF ($\pi_T = 0.1$) as a function of C (again, use a logarithmic scale for the x-axis). What do you observe? Does the regularization coefficient significantly affect the results for one or both metrics (remember that, for SVM, low values of C imply strong regularization, while large values of C imply weak regularization)? Are the scores well calibrated for the target application? What can we conclude on linear SVM? How does it perform compared to other linear models? Repeat the analysis with centered data. Are the result significantly different?

$K = 0 \quad K = \sqrt{\xi} = \sqrt{0} = 0$

2

We now consider the polynomial kernel. For simplicity, we consider only the kernel with $d = 2, c = 1$ (but better results may be possible with different configurations), and we set $\xi = 0$, since the kernel already implicitly accounts for the bias term (due to $c = 1$). We also consider only the original, non-centered features (again, different pre-processing strategies may lead to better results). Train the model with different values of C, and compare the results in terms of minDCF and actDCF. What do you observe with quadratic models? In light of the characteristics of the dataset and of the classifier, are the results consistent with previous models (logistic regression and MVG models) in terms of minDCF? What about actDCF?

$\xi = 1 \quad K = 1$

3

For RBF kernel we need to optimize both γ and C (since the RBF kernel does not implicitly account for the bias term we set $\xi = 1$). We adopt a grid search approach, i.e., we consider different values of γ and different values of C, and try all possible combinations. For γ we suggest you analyze values $\gamma \in [e^{-4}, e^{-3}, e^{-2}, e^{-1}]$, while for C, to avoid excessive time but obtain a good coverage of possible good values we suggest log-spaced values `numpy.logspace(-3, 2, 11)` (of course, you are free to experiment with other values if you so wish). Train all models obtained by combining the values of γ and of C. Plot minDCF and actDCF as a function of C, with a different line for each value of γ (i.e., four lines for minDCF and four lines for actDCF). Analyze the results. Are there values of γ and C that provide better results? Are the scores well calibrated? How the result compare to previous models? Are there characteristics of the dataset that can be better captured by RBF kernels?

Optional → FARLO IN CASO PER AGGIUNGERLO AL REPORT

Consider again the polynomial kernel, but with $d = 4, c = 1, \xi = 0$. Train the model with different values of C (use again `numpy.logspace(-5, 0, 11)`, and compare the results in terms of minDCF

UTILE SE C'È UNA RELAZIONE POLINOMIALE

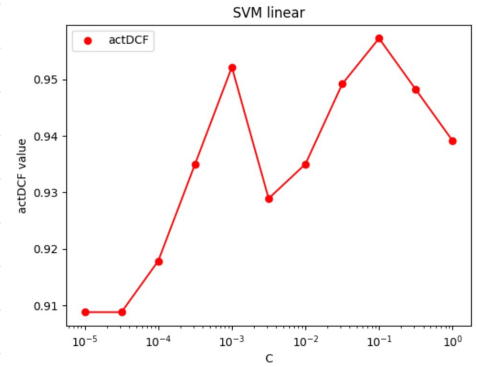
and actDCF. What do you observe with quadratic models? Can you explain the better results in terms of the characteristics of the dataset? To answer, consider only the last two features of each sample (look at the scatter plots) $\boldsymbol{y}_i = \boldsymbol{x}_{i,[4:6]}$. Consider how these features would be transformed by a simple degree 2 kernel that maps each sample to $\boldsymbol{y}_i \rightarrow z_i = \boldsymbol{y}_{i,[0:1]}\boldsymbol{y}_{i,[1:2]}$ (suggestion: draw few samples on paper, one per cluster). Then consider which kind of separation surfaces would be required to separate the z_i 1-D feature vectors — remember that in 1-D linear rules correspond to a sample being on either side of a threshold, quadratic rules correspond to inequalities that involve the sample being inside a (possibly empty) interval, and so on.

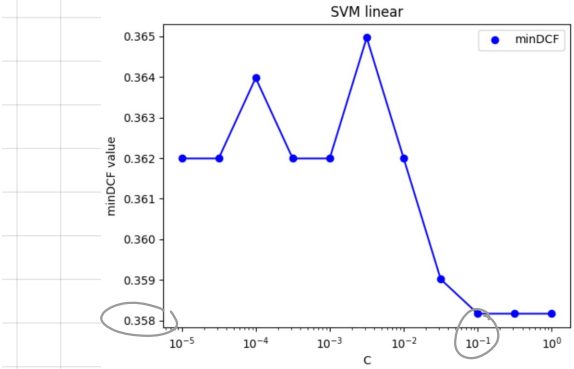
1)

SVM
{'K': 1.0, 'C': 1e-05}
error rate 9.15
minDCF 0.36199116743471577
actDCF 0.9087941628264209
primal value 0.039286223943141046
dual value 0.03928622394313975
duality gap 1.2975731600306517e-15
{'K': 1.0, 'C': 3.1622776601683795e-05}
error rate 9.15
minDCF 0.36199116743471577
actDCF 0.9087941628264209
primal value 0.11935334583814561
dual value 0.11935334583814008
duality gap 5.523359547510154e-15
{'K': 1.0, 'C': 0.0001}
error rate 9.2
minDCF 0.36397529441884274
actDCF 0.9178667434715821
primal value 0.32901700908015585
dual value 0.32901700908014475
duality gap 1.1102230246251565e-14
{'K': 1.0, 'C': 0.00031622776601683794}
error rate 9.25
minDCF 0.36199116743471577
actDCF 0.9350198412698413
primal value 0.7734635850050467
dual value 0.7734635850038079
duality gap 1.2388978731792122e-12
{'K': 1.0, 'C': 0.001}
error rate 9.3
minDCF 0.36199116743471577
actDCF 0.9521729390681003
primal value 1.7879017276169402
dual value 1.787901726985842
duality gap 6.310982847423929e-10
{'K': 1.0, 'C': 0.0031622776601683794}
error rate 9.35
minDCF 0.3649673579109063
actDCF 0.9289234511008704
primal value 4.357344778910858
dual value 4.3573447728171155
duality gap 6.093742399571056e-09
{'K': 1.0, 'C': 0.01}
error rate 9.25
minDCF 0.36199116743471577
actDCF 0.9350198412698413
primal value 11.432594525231403
dual value 11.432594515181425
duality gap 1.0049978627080236e-08
{'K': 1.0, 'C': 0.03162277660168379}
error rate 9.15
minDCF 0.3590149769585253
actDCF 0.9491967485919098
primal value 32.638393859985854
dual value 32.6383937664235
duality gap 9.35623560849308e-08
{'K': 1.0, 'C': 0.1}
error rate 9.15
minDCF 0.3581669226830517
actDCF 0.9572772657450076
primal value 98.55103667730022
dual value 98.55103626953144
duality gap 4.077687805192909e-07
{'K': 1.0, 'C': 0.31622776601683794}
error rate 9.1
minDCF 0.3581669226830517
actDCF 0.9482046850998463
primal value 306.21343912645773
dual value 306.2134355660369
duality gap 3.5604208505901624e-06
{'K': 1.0, 'C': 1.0}
error rate 9.049999999999999
minDCF 0.3581669226830517
actDCF 0.939132104454685
primal value 962.5958798209837
dual value 962.5958646253371
duality gap 1.519564659702155e-05

Nel tuo caso, sembra che il "duality gap" sia molto piccolo per tutti i valori di C, il che indica che l'ottimizzazione SVM è stata risolta con precisione. Inoltre, noterai che il "duality gap" tende a crescere con l'aumentare del parametro C. Questo è normale, poiché un valore di C più grande corrisponde a un problema di ottimizzazione più difficile (poiché si sta cercando di classificare correttamente più punti), che può portare a un "duality gap" più grande.

Inoltre, i valori di errore, minDCF e actDCF ti danno un'idea delle prestazioni del tuo modello SVM. L'errore rate ti dice la percentuale di punti mal classificati. Il minDCF (minimo Detection Cost Function) è una misura di prestazione che combina sia la probabilità di falsi positivi che quella di falsi negativi. L'actDCF (actual Detection Cost Function) è una stima del costo effettivo basato sulla matrice di confusione. Idealmente, vorresti che questi valori fossero il più bassi possibile, indicando un modello di classificazione di alta qualità





MINDCF = 0.358

2) I modelli quadratici, come quelli utilizzati nelle SVM con kernel polinomiale di grado 2 nel tuo codice, sono un tipo di modello di apprendimento automatico che può catturare relazioni non lineari tra le caratteristiche. Questi modelli mappano le caratteristiche di input in uno spazio di caratteristiche di dimensione superiore dove i dati possono essere separati da un iperpiano, che in questo caso è una curva parabolica nel caso bidimensionale

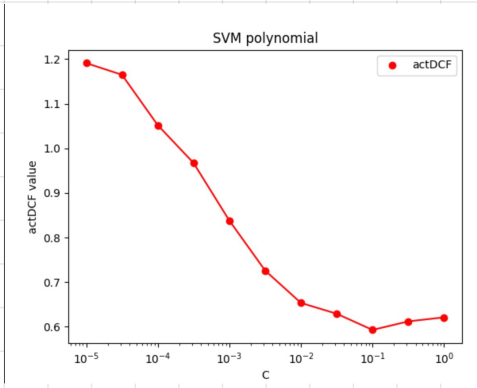
VARIANDO C PUOI VEDERE COME SI COMPORTA UN MODELLO PIÙ COMPLESSO, VISTO CHE C PIÙ ALTE FANNO AUMENTARE LA COMPLESSITÀ DEL MODELLO. UNA C PIÙ PICCOLA POTREBBE NON FAR ADATTARE IL MODELLO AI DATI

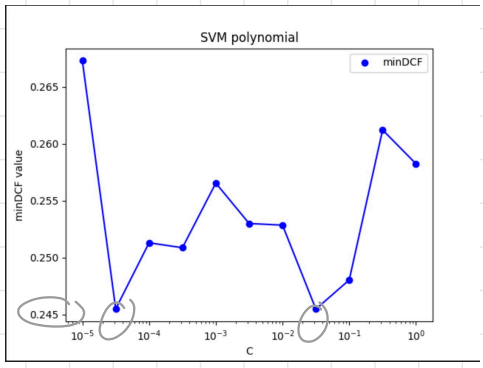
DIFFERENZE TRA I KERNEL → PUÒ ESSERE PIÙ EFFICACE SE C'È UNA RELAZIONE POLINOMIALE TRA LE CARATTERISTICHE

- KERNEL POLINOMIALE: MAPPA I DATI IN UNO SPAZIO DI DIMENSIONE SUPERIORE UTILIZZANDO UNA FUNZIONE POLINOMIALE. UTILE SE C'È RELAZIONE POLINOMIALE
- KERNEL RBF: KERNEL GAUSSIANO, MAPPA IN UNO SPAZIO DI DIM INFINITA $e^{-\gamma \|x-y\|^2}$ UTILE SE NON C'È UNA FORMA SPECIFICA DI RELAZIONE.

γ → SCALA

$\|x-y\|^2$ → DISTANZA EUCLIDEA TRA I DUE VETTORI DI INPUT

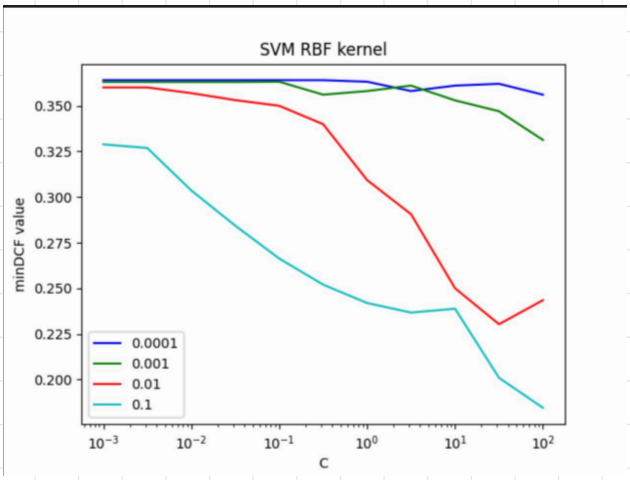


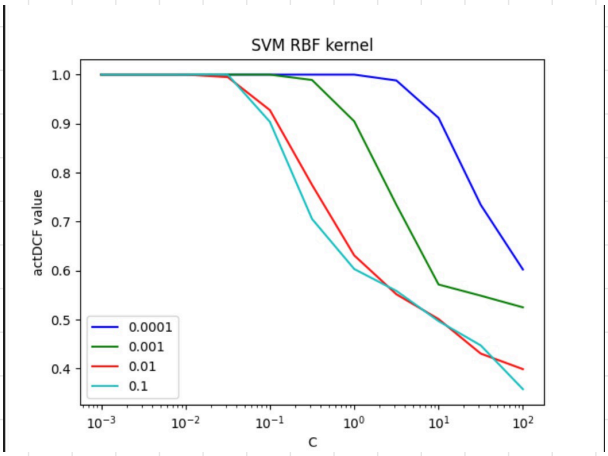


MINDCF = 0.245

QUESTI SONO I MIGLIORI RISULTATI SIMILI A QUELLI EXPANDED DEL LR DOBBIAMO PRENDERE PER MVG QUELLI CON LA STESSA PRIOR 0.1 QUELLI HANNO VALORI SIMILI.

3)





E' normale che per valori molto piccoli di γ eC, l'actDCF possa essere molto grande rispetto agli altri. Questo comportamento può essere spiegato dal modo in cui γ e C influenzano il modello SVM con kernel RBF.

γ (Parametro del Kernel RBF): γ controlla l'influenza di un singolo punto di addestramento. Un valore molto piccolo di γ significa che l'influenza di ogni punto di addestramento è molto ampia, il che può portare a un modello troppo semplice (underfitting), incapace di catturare la complessità dei dati. Un valore molto alto di γ significa che l'influenza di ogni punto di addestramento è molto ristretta, il che può portare a un modello troppo complesso (overfitting), che si adatta troppo ai dati di addestramento e non generalizza bene ai dati di test.

C (Parametro di Penalizzazione): C controlla il trade-off tra la massimizzazione del margine e la minimizzazione dell'errore di classificazione. Un valore molto piccolo di C significa che il modello permette molti errori di classificazione, preferendo un margine più ampio, il che può portare a underfitting. Un valore molto grande di C significa che il modello penalizza severamente gli errori di classificazione, preferendo un margine più stretto, il che può portare a overfitting.

Quando entrambi i parametri γ e C sono molto piccoli, il modello tende a underfitting. In questa situazione, il classificatore non riesce a separare correttamente le classi nei dati di test, portando a elevate probabilità di mancata rilevazione (miss) e falsi allarmi. Di conseguenza, il valore di actDCF può diventare molto alto.

PERCHÉ minDCF SEMBRA ESSERE NELLA NORMA MENTRE ACTDCF NO

è possibile che per valori molto piccoli di γ e C, il minDCF sia nella norma mentre l'actDCF è molto alto. Questo comportamento può essere spiegato dalla differenza tra come vengono calcolati minDCF e actDCF e come i parametri del modello influenzano queste metriche.

Differenza tra minDCF e actDCF

- **minDCF (Minimum Detection Cost Function)**:
- È calcolato scegliendo la soglia di decisione ottimale che minimizza il costo di rilevamento.
- Rappresenta il miglior costo teorico che il sistema può ottenere, indipendentemente dalla soglia di decisione attuale.
- La soglia di decisione è ottimizzata per bilanciare le probabilità di mancata rilevazione (miss) e falso allarme (false alarm).
- **actDCF (Actual Detection Cost Function)**:
- È calcolato utilizzando una soglia di decisione specifica, che può non essere ottimale.
- Rappresenta il costo di rilevamento effettivo del sistema con la soglia di decisione utilizzata durante la classificazione.

Implicazioni dei Parametri

Valori piccoli di γ

γ:

1. • Portano a un modello che generalizza eccessivamente, producendo una decision boundary molto liscia.
2. • Potrebbe non catturare la complessità dei dati, portando a un modello che non riesce a separare correttamente le classi.
3. **Valori piccoli di** C

C:

4. • Permettono molti errori di classificazione, preferendo un margine più ampio.
5. • Questo porta a un modello che penalizza meno gli errori, potenzialmente causando underfitting.

Situazione Tipica

Quando γ e C sono molto piccoli, il modello SVM potrebbe non essere in grado di separare bene le classi nei dati di addestramento e test. Tuttavia, il minDCF è calcolato ottimizzando la soglia di decisione, quindi può ancora risultare nella norma se esiste una soglia che bilancia le probabilità di mancata rilevazione e falso allarme. In altre parole, il minDCF rappresenta il miglior costo possibile che il modello può ottenere, mentre l'actDCF rappresenta il costo effettivo con la soglia di decisione utilizzata.

Esempio di Comportamento

Immagina un dataset con due classi parzialmente sovrapposte. Con valori molto piccoli di γ e C, il modello SVM potrebbe produrre una decision boundary che non separa bene le classi. Tuttavia, esiste una soglia di decisione ottimale che minimizza il costo di rilevamento, portando a un minDCF accettabile. Allo stesso tempo, la soglia di decisione utilizzata effettivamente potrebbe non essere ottimale, portando a un actDCF elevato.

Conclusione

È normale che il minDCF sia nella norma mentre l'actDCF è elevato per valori molto piccoli di γ e C. Questo perché il minDCF ottimizza la soglia di decisione per ottenere il miglior costo possibile, mentre l'actDCF riflette il costo effettivo con la soglia di decisione utilizzata durante la classificazione. Per migliorare l'actDCF, è importante ottimizzare sia i parametri γ e C che la soglia di decisione. Tecniche come Grid Search con validazione incrociata possono aiutare a trovare i valori ottimali di questi parametri e migliorare le prestazioni del modello.