assignment_07_solution

April 2, 2023

Please fill in your name and that of your teammate.

You:

Teammate:

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
sns.set(rc={'figure.figsize':(8,6)}, style="whitegrid")
```

1 Introduction

Welcome to the seventh lab. After learning about SVMs last week, we finally introduce the *kernel trick* and make them capable of tackling nonlinear data. We also introduced more generally the concept of *function mapping* and learned a bit about word embeddings.

1.0.1 How to pass the lab?

Below you find the exercise questions. Each question awarding points is numbered and states the number of points like this: **[0pt]**. To answer a question, fill the cell below with your answer (markdown for text, code for implementation). Incorrect or incomplete answers are in principle worth 0 points: to assign partial reward is only up to teacher discretion. Over-complete answers do not award extra points (though they are appreciated and will be kept under consideration). Save your work frequently! (`ctrl+s`)

You need at least 16 points (out of 24 available) to pass (66%).

2 1. Function Mappings

1.1 [3pt] Give an *original* example for each of the following concepts (i.e. not one that you find in the slides!):

1. Mapping from an example data type to a decision space
2. Inverse mapping
3. Mapping from the example data to two destination feature spaces
4. Mapping from the two feature spaces above to a decision space

For example (here is what we presented in the slides):

1. Given a picture, decide if it represents an apple or an orange. Original space: $64 \times 64 \times 3$ images; destination space: {apple, orange}.
2. Given the label apple, map it to a $64 \times 64 \times 3$ picture of the fruit.
3. Map Φ_1 goes from the picture to an estimate of average color; map Φ_2 goes from the picture to a fruit width measured in pixels.
4. Map Φ_3 goes from the two features of estimated color and fruit width, to the decision space of classifying the fruit as apple or orange.

The example above is by definition already a correct solution.

1.2 [1pt] Explain advantages and disadvantages of Bag Of Words versus Word Embedding. Bag of Words produce a higher-dimensional feature map (size of the dictionary) that is very sparse (each text only uses a subset of the dictionary). Word Embeddings produce a smaller feature space, which is dense.
More importantly, BoW only uses tallying of the used words (counts the number of occurrences), while WE maintains the distance between words in the text, encoding words that tend to appear often together with closer vectors in feature space. This allows deduction and reasoning of word relationships based on vector similarity (distance and angle).

1.3 [2pt] Refer to the graph exemplifying Word Embedding in the slides, and its explanation. (i) What does it mean that the point representing Paris is close to the point representing Berlin? (ii) Why is the point for Paris closer to the point for France than to the point representing Italy?

(i) It means that Paris and Berlin refer to two similar concepts, that often appear together in the text corpus. (ii) It means that Paris tends to appear more often closer to France than to Italy in the text corpus.

3 2. Kernels theory

2.1 [1pt] Write the definition of kernel function (use latex).

$$k(x, y) = \langle \phi(x), \phi(y) \rangle$$

2.2 [2pt] Explain the kernel trick in English. The kernel trick states that the kernel function between two points is equivalent to computing the dot product between the mapping of the two data points into RKHS. This is useful because it allows us to use all the positive properties of RKHS while avoiding working with the mapping function directly, which can potentially be complex. The actual formulation of the kernel can be any form that produces a Gram matrix that is symmetric and positive semi-definite.

2.3 [1pt] Explain in English the required properties of a Mercer kernel.

- Symmetry: the order of the parameters to the kernel does not matter.
- Positive definiteness: the Gram matrix of the kernel is positive semi-definite for any finite subset of the input space.

2.4 [1pt] Calculate by hand the linear kernel on points $\{[2,4],[1,-2]\}$. The linear kernel is simply the dot product:

$$k([2,4],[1,-2]) = [2,4]^{\mathrm{T}} \cdot [1,-2] = 2 - 8 = -6$$

2.5 [1pt] How do you compute the entry of the Gram matrix for row i and column j for a Gaussian kernel?

$$K_{ij} = k(x_i, x_j) = \exp(-\gamma \cdot \|x_1 - x_2\|^2)$$

2.6 [2pt] Explain why does the Perceptron work with non-linearly separable data using Kernelization. Do you think Linear Regression would work with Kernelization? Explain your reasoning. Kernelization maps non-linearly separable data to a space where the data is hopefully linearly separable. At that point, the Perceptron is able to do the linear separation. The same works with any linear algorithm, including Linear Regression: once the Kernelization takes care of nonlinearities, the linear algorithm can do its job without problems.

4 3. Kernels in practice

For simplicity, let's use once again a two-species adaptation of the Iris dataset. You can copy the code from the last assignment. This time though, to make it harder for linear classifiers let's separate the "central" species from the other two. This means that you should set label `versicolor` rather than `setosa` as class `-1`. I suggest you un-comment the `pairplots` to verify it works.
NOTE: all recommendation on how to handle and prepare the data from the past assignment(s) still hold. As do the warnings that using the wrong data sets will **invalidate the whole answer**.

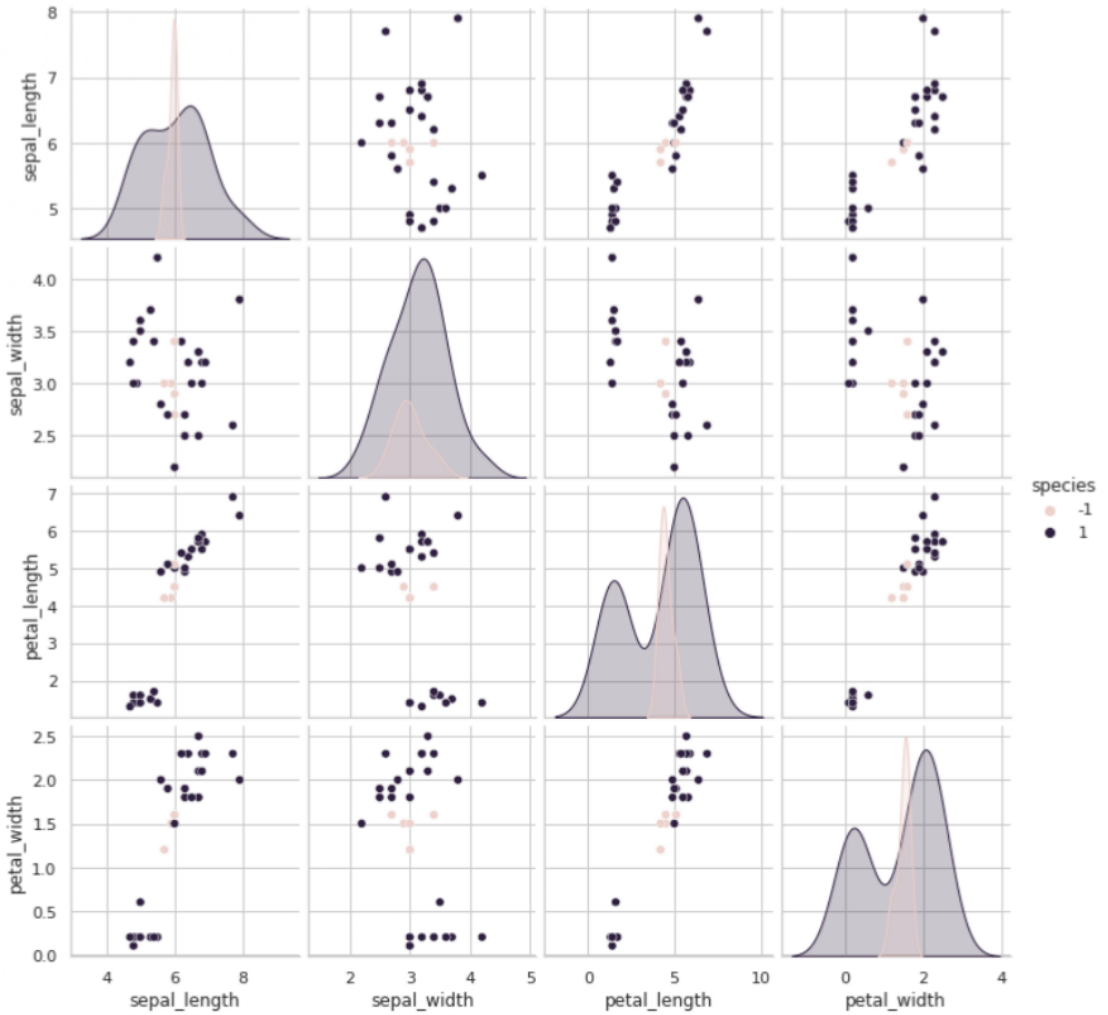
[2]:
```
df = sns.load_dataset('iris')
# sns.pairplot(df, hue='species')

df.loc[df['species'] == 'versicolor', 'species'] = -1
df.loc[df['species'] != -1, 'species'] = 1
df['species'] = pd.to_numeric(df['species'])
print(df.dtypes)
# sns.pairplot(df, hue='species')

from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=0.2) # 80-20 split
sns.pairplot(test, hue='species')
```

```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species           int64
dtype: object
```

[2]: `<seaborn.axisgrid.PairGrid at 0x7f68b05b2130>`



3.1 [2pt] Train an SVM with linear kernel on the Iris data using Scikit-learn (this time you are required to use class `SVC`). Then do the same using a Gaussian kernel (still `SVC`) and compare the performance using its method `score()`.

- Remember to prepare inputs/labels for Scikit-learn; again the last assignment should help.
- Calling the method `score()` on the trained model just does the prediction and returns the percentage of correct answers. It is a useful function to learn to quickly check if your model is working.
- You expect the linear kernel to perform poorly. If the performance is close to the Gaussian kernel, it is possible that the test set was by chance not homogeneous. You can verify that by doing a pairplot on the test set, and if so just run the data loading and preparation again.
- No need to find an optimal value for `C` but pass it explicitly.

[3]:
```
from sklearn.svm import SVC

x_train = train.iloc[:,:-1]
```

```
y_train = train['species']
x_test = test.iloc[:,:-1]
y_test = test['species']

lin = SVC(kernel='linear', C=1).fit(x_train, y_train)
print(lin.score(x_test, y_test))

gau = SVC(kernel='rbf', C=1).fit(x_train, y_train)
print(gau.score(x_test, y_test))
```

```
0.7
0.9333333333333333
```

3.2 [2pt] Write a Python function that takes two data points and a value for `gamma` as input, and returns the Gaussian kernel of the points.

[4]:
```
kern = lambda x, y, gamma: np.exp(- gamma * np.linalg.norm(x - y)**2)
```

3.3 [3pt] Write a Python function that takes two dataset (and a gamma) and returns their Gram matrix for a Gaussian kernel.

- You need two datasets because you need to compute the *train* matrix between the train and itself, but the *test* matrix between the test and the train.
- Simplest method:
  - Create a return matrix, initially empty, shaped size_of_A times size_of_B, with dtype 'float64'
  - Run two loops with indices (i, j) in ranges up to size_of_A and size_of_B
  - Compute the kernel between row i in A and row j in B, and place it in the return matrix at row i column j
- Careful with Pandas' `iterrows()`, as the "index" it returns is the DataFrame index (i.e. for use with `loc[]`), not the ordinal index (i.e. for `iloc[]`).
- Generating the matrix automatically is harder, as there is no straightforward way to compute an `outer` in numpy or pandas with a custom function.
- One way is to use `column_stack` https://stackoverflow.com/a/21759340 then apply the kernel defined above.
- Another is to use `ufunc.outer` http://folk.uio.no/inf3330/scripting/doc/python/NumPy/Numeric/numpy-7.html which is only defined for Universal Functions (`ufuncs`). Look at the examples for `outer`, you can re-implement the function above starting with `np.subtract.outer(A, B)`, which generates the matrix (but check the shape!), then you can run the other operations using broadcast. Both outers and universal functions are super useful, it's worth the effort of learning them, more [here].
- `pandas.apply()` along rows is also an option you should be able to consider with by now. The function name for `-` is `np.subtract` (which is an `ufunc`, see above).

Above all, remember the first rule of a good BDD engineer: red, green, refactor! First make it work, then make it better ;) complex solutions are as good as bonus questions here.

Also, know that a common default value for gamma is one over the number of features.

[5]:
```
# This is not the most efficient version (by far). But should be readable by␣
 ↪everyone.
# Is your answer better?
def gram(dset_1, dset_2, gamma=None):
    npoints_1, nfeats_1 = dset_1.shape
    npoints_2, nfeats_2 = dset_2.shape
    assert nfeats_1 == nfeats_2 # always check for consistency
    if gamma is None: gamma = 1/nfeats_1
    gmat = np.empty([npoints_1, npoints_2], dtype='float64')
    for r in range(npoints_1):
        for c in range(npoints_2):
            # hint: which operations you need here and which can you broadcast?
            gmat[r][c] = kern(dset_1.iloc[r], dset_2.iloc[c], gamma)
    return gmat
```

3.4 [2pt] Compute the Gram matrix on the inputs of your datasets. Then train a new SVM, same settings as before with linear kernel, but this time using the Gram matrix('s rows) as the inputs. Print the `score` performance of this new SVM.

- With an 80-20 split you are looking at a 120×120 shape for the train, and 30×120 for the test

[6]:
```
x_train_gram = gram(x_train, x_train)
x_test_gram = gram(x_test, x_train)
print(x_train_gram.shape, x_test_gram.shape)
```

```
(120, 120) (30, 120)
```

[7]:
```
gauss_plus_lin = SVC(kernel='linear').fit(x_train_gram, y_train)
print(gauss_plus_lin.score(x_test_gram, y_test))
```

```
0.9666666666666667
```

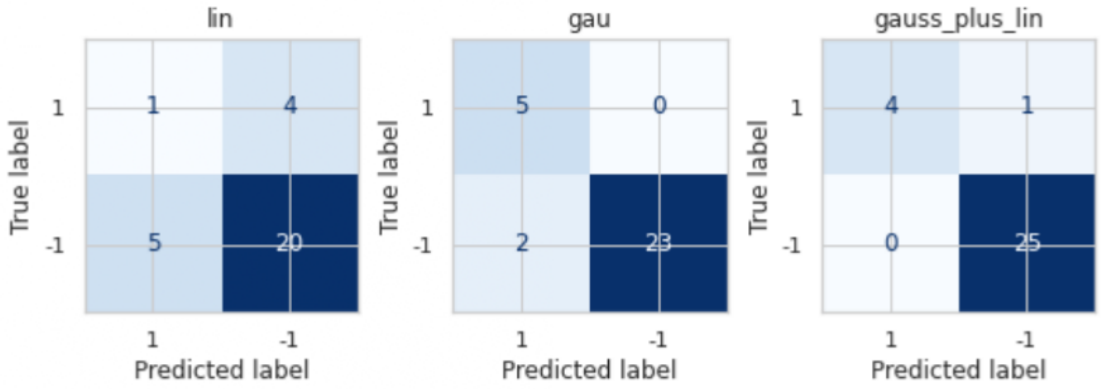
3.5 [1pt] Plot the confusion matrix for the three SVMs you trained in the past questions.

- Let's learn a convenient and easy function for this common, very useful metric: `ConfusionMatrixDisplay.from_estimator` [link here].
- So far we saw explicitly 4 cells: true and false positives, true and false negatives. More generally the confusion matrix can be scaled to any number of classes by having the correct labels on the rows, and the predictions on the columns. Errors will be outside the diagonal.
- You can use the `normalize` option to get percentages if you like. Which setting do you find most informative?
- It's easier if you write a `for` loop over the three models you trained in the previous questions – just make sure you gave them different names. Also careful as one takes a Gram matrix as input ;)

[8]:
```
# from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
class_names = y_test.unique()
# It's about time we learn to use subplots, especially for small things
fig, axes = plt.subplots(1,3)
fig.tight_layout()

# Here's another trick: you can list the _names_ of the variables,
# then use their name on the title, and fetch their values using `eval()`.
# This is not optimal here, but I wanted to expose you to this technique,
# as it opens the door to metaprogramming (you could _build_ the name-strings!)
for model_name, ax in zip(["lin", "gau", "gauss_plus_lin"], axes):
    if model_name == "gauss_plus_lin":
        x_dset = x_test_gram
    else:
        x_dset = x_test
    ConfusionMatrixDisplay.from_estimator(
        eval(model_name), x_dset, y_test,
        ax=ax,
        display_labels=class_names,
        colorbar=False,
        cmap=plt.cm.Blues,
        normalize=None)

    ax.set_title(model_name)
```



5 At the end of the exercise

Bonus question with no points! Answering this will have no influence on your scoring, not at the assignment and not towards the exam score – really feel free to ignore it with no consequence. But solving it will reward you with skills that will make the next lectures easier, give you real applications, and will be good practice towards the exam.

The solution for this questions will not be included in the regular lab solutions pdf, but you are welcome to open a discussion on the Moodle: we will support your addressing it, and you may meet other students that choose to solve this, and find a teammate for the next assignment that is willing to do things for fun and not only for score :)

BONUS [ZERO pt] Use a contour plot to show the classification boundaries of your SVMs. [link here]

BONUS [ZERO pt] Learn to search for the best values for `gamma` and `C` [link here] . NOTE: this is extremely valuable experience for when (not even if!) you will need a SVM for a real application.

BONUS [ZERO pt] Generate points to run through with a regression algorithm like Linear Regression from our earlier exercises. This time start from a nonlinear equation (e.g. x^2), and add noise as usual. Then try your hand with SVR with a (linear or) nonlinear kernel, which is equivalent to running Linear Regression on the Gram matrix (yet another name: Kernel Ridge Regression) [link here].

5.0.1 Final considerations

- I once read a quote that restricting calculus to linear functions is like restricting biology to the study of great apes (help tracking its origin would be welcome). We start from linearity because it's easier to study; the real world is rarely so kind, so learning adaptations such as the kernel trick is simply invaluable.
- Trying (scikit-learn) Naïve Bayes or Linear Discriminant Analysis on the Gram matrices would take you just a minute and be invaluable experience. For example, I wouldn't be surprised if LDA performed better than NB (think: why?). But if we had a very large dataset, the Gram matrix would become too large for LDA to handle (remember it does not scale well on the number of features).
- **[IMPORTANT]** If you want to gain first-hand experience in tools you can actually use in the real world, consider submitting on the bonus questions from this point on, as I am switching the topic from "topics for curiosity" to "actual deployed value".