Mathematics of Machine Learning

Philipp Christian Petersen[*]

April 18, 2022

Contents

[*]PP can be found in room 7.39 in the Kolingasse 14-16, 1090 Vienna, unless there is a lockdown. In that case, he will hide at an undisclosed location, but will still be very willing to read emails sent to Philipp.Petersen@univie.ac.at.

0 Introduction

This lecture is about:

1. Language of ML: What is classification, regression, ranking, clustering, dimensionality reduction, supervised and unsupervised learning, generalisation, overfitting.
2. PAC Theory: PAC Learning model, finite hypothesis sets, consistent and inconsistent problems, deterministic and agnostic learning.
3. Rademacher complexity and VC dimension: generalization bounds for Rademacher, Growth function, Connection to Rademacher compl., VC dimension, VC dimension based upper bounds, lower bounds on generalization.
4. Model Selection: Bias Variance trade-off, Structural Risk minimisation, Cross validation, regularisation.
5. Support Vector Machines: generalisation bounds, margin theory/margin based generalization bounds.
6. Kernel Methods: Reproducing Kernel Hilbert spaces, Representer Theorem, kernel SVM, generalisation bounds for kernel based methods
7. Neural Networks (Mostly shallow)
8. Clustering: k-means, Lloyds algorithm, Ncut, Cheeger cut, spectral clustering.
9. Dimensionality Reduction: PCA, diffusion maps, Johnson Lindenstrauss lemma.

Literature:

- Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of machine learning. MIT press, 2018. `https://cs.nyu.edu/~mohri/mlbook/`
- Shalev-Shwartz, Shai, and Shai Ben-David. Understanding machine learning: From theory to algorithms. Cambridge university press, 2014. `https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/`
- Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media, 2009 `https://web.stanford.edu/~hastie/ElemStatLearn/`

Lecture notes: You are reading the lecture notes to the course right now. They are being developed during the course. If you find things than can be improved, please contact Philipp Petersen to give feedback. This is highly appreciated.

Required prior knowledge: This is an applied math course. Therefore it will often touch on many different mathematical fields. Such as harmonic analysis, graph theory, random matrix theory, etc. Students are not required to know about these issues beforehand. But a certain willingness to look up concepts from time to time is necessary.

To enjoy this course, students should have heard an introductory course on *probability theory* and *linear algebra*. Most of the examples, as well as the challenges will require rudimentary knowledge of *Python*. Towards the end, a basic knowledge of functional analysis is helpful, but not required.

Assessment criteria: There will be an oral exam at the end of the lecture.

In addition, we will have challenges where you will solve machine learning problems in Python. You need to form groups for this. You need to beat me to pass (This will be very easy, because I will publish my approach and my solutions will typically not be very sophisticated). The winning team and the team with the most creative solution will receive prices. There will be at least three challenges this year.

Note that your results for the challenges need to be handed in before the given deadline. **There are no exceptions to this rule.** You will have plenty of time to complete these, so it may make sense to prepare one submission as a backup and then fine tune later.

1 Lecture 1 – Example and Language of Machine Learning

A cheesy lecture on machine learning would probably start by claiming that machine learning is revolutionary and constitutes a completely new paradigm for science and mathematics. One may even say a new world of unknown and exciting terrain with uncountable possibilities.

We instead show that machine learning can quite literally show us new worlds.

1.1 A new world (literaly)

[1]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.style as style
style.use('seaborn')
import numpy as np
from scipy import signal
from scipy.fftpack import fft, ifft
from scipy.signal.windows import gaussian
```

We import a data set of light curves of stars recorded from the Kepler telescope. These can be found online at (https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data). We print the first five lines of the data set to get a feeling what is going on.

[2]:
```
import pandas as pd
data = pd.read_csv("exoTrain.csv")
# Preview the first 5 lines of the loaded data
data.head()
```

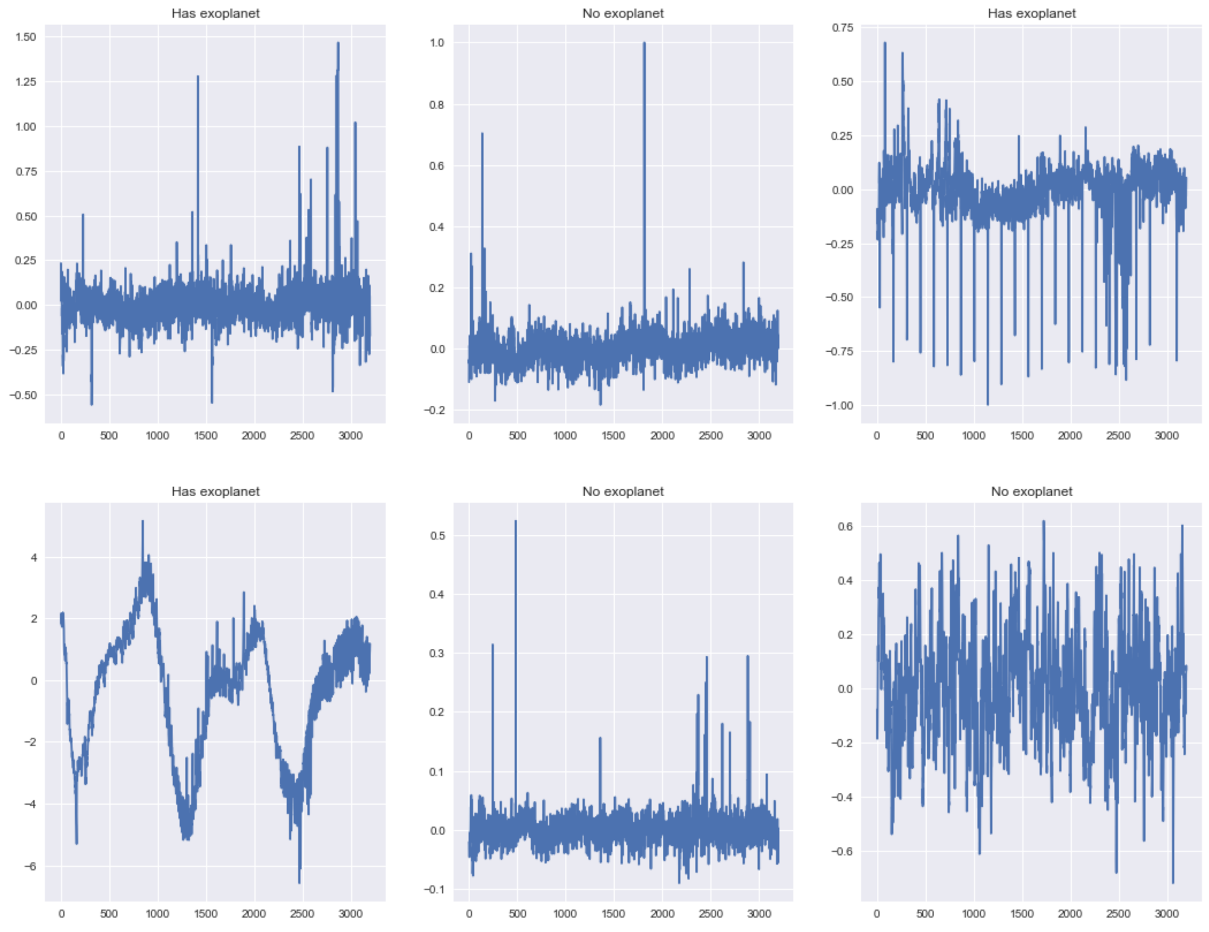
[2]:
```
   LABEL   FLUX.1   FLUX.2   FLUX.3   FLUX.4   FLUX.5   FLUX.6  FLUX.7  \
0      2    93.85    83.81    20.10   -26.98   -39.56  -124.71 -135.18
1      2   -38.88   -33.83   -58.54   -40.09   -79.31   -72.81  -86.55
2      2   532.64   535.92   513.73   496.92   456.45   466.00  464.50
3      2   326.52   347.39   302.35   298.13   317.74   312.70  322.33
4      2 -1107.21 -1112.59 -1118.95 -1095.10 -1057.55 -1034.48 -998.34
[5 rows x 3198 columns]
```

The columns are the intensities of the light at different positions in time. The label is 2 if some astrophysicists has claimed that this planet has an exoplanet and 1 if they claimed it has none. We will plot a couple of these curves to get a good understanding what is going on.

[3]:
```
plt.figure(figsize = (10, 8))

fig = plt.figure(figsize=(18,14))
ax = fig.add_subplot(231)
plt.plot(data.values[6, 1:]/np.max(np.abs(data.values[69, 1:])))
plt.title('Has exoplanet')
ax = fig.add_subplot(232)
plt.plot(data.values[2003, 1:]/np.max(np.abs(data.values[2003, 1:])))
plt.title('No exoplanet')
ax = fig.add_subplot(233)
plt.plot(data.values[1, 1:]/np.max(np.abs(data.values[1, 1:])))
plt.title('Has exoplanet')
ax = fig.add_subplot(234)
plt.plot(data.values[13, 1:]/np.max(np.abs(data.values[69, 1:])))
plt.title('Has exoplanet')
ax = fig.add_subplot(235)
plt.plot(data.values[75, 1:]/np.max(np.abs(data.values[2003, 1:])))
plt.title('No exoplanet')
ax = fig.add_subplot(236)
plt.plot(data.values[77, 1:]/np.max(np.abs(data.values[1, 1:])))
plt.title('No exoplanet')
```

[3]:
```
Text(0.5, 1.0, 'No exoplanet')
```

```
<Figure size 720x576 with 0 Axes>
```



Stars with exoplanets often have periodically occuring sharp drops in light intensity. We do not know if it is the only indication, though. Since we also not trained in astrophysics, we should not overanalyse this.

Maybe there is another obvious way of differentiating between stars with exoplanets and stars. We shall start some exploratory data analysis. This consists of looking at certain statistical aspects of the data set:

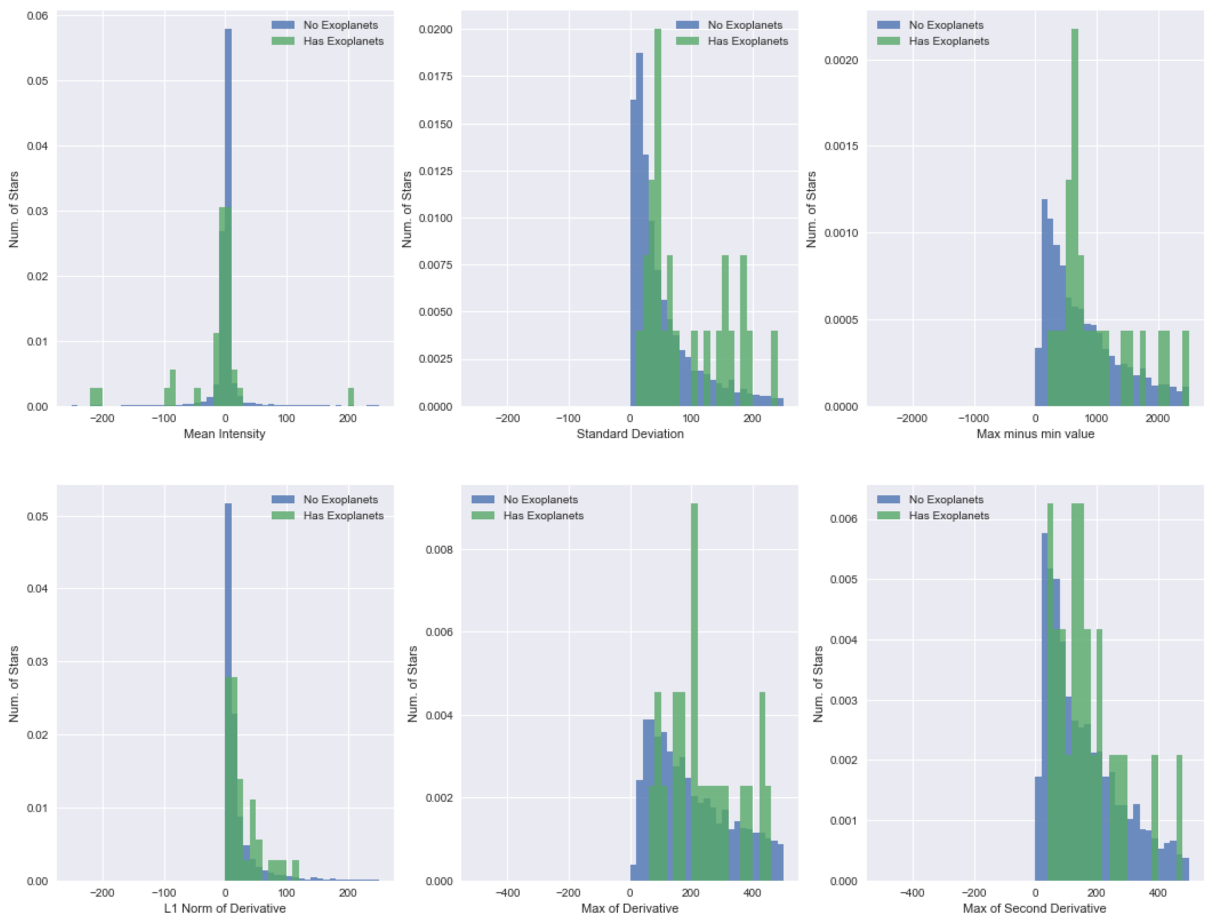
[4]:
```
LightCurves = data.values[:, 1:]


ex_labels = data.values[:, 0]
```

```
print('In the data set there are: '  + str(np.sum(ex_labels==1)) + ' Stars without exoplanets.')
print('In the data set there are: '  + str(np.sum(ex_labels==2)) + ' Stars without exoplanets.')

fig = plt.figure(figsize=(18,14))

means1 = LightCurves[ex_labels==1].mean(axis=1)
means2 = LightCurves[ex_labels==2].mean(axis=1)

ax = fig.add_subplot(231)

ax.hist(means1,alpha=0.8,bins=50,density=True,range=(-250,250))
ax.hist(means2,alpha=0.8,bins=50,density=True,range=(-250,250))
ax.legend(['No Exoplanets', 'Has Exoplanets'])
ax.set_xlabel('Mean Intensity')
ax.set_ylabel('Num. of Stars')

std1 = LightCurves[ex_labels==1].std(axis=1)
std2 = LightCurves[ex_labels==2].std(axis=1)

ax = fig.add_subplot(232)

ax.hist(std1,alpha=0.8,bins=50,density=True,range=(-250,250))
ax.hist(std2,alpha=0.8,bins=50,density=True,range=(-250,250))
ax.legend(['No Exoplanets', 'Has Exoplanets'])
ax.set_xlabel('Standard Deviation')
ax.set_ylabel('Num. of Stars')

spread1 = LightCurves[ex_labels==1].max(axis=1) - LightCurves[ex_labels==1].min(axis=1)
spread2 = LightCurves[ex_labels==2].max(axis=1) - LightCurves[ex_labels==2].min(axis=1)

ax = fig.add_subplot(233)

ax.hist(spread1,alpha=0.8,bins=50,density=True,range=(-2500,2500))
ax.hist(spread2,alpha=0.8,bins=50,density=True,range=(-2500,2500))
ax.legend(['No Exoplanets', 'Has Exoplanets'])
ax.set_xlabel('Max minus min value')
ax.set_ylabel('Num. of Stars')

Derivative = np.abs(np.gradient(LightCurves[ex_labels==1], axis = 1)).mean(axis=1)
Derivative2 = np.abs(np.gradient(LightCurves[ex_labels==2], axis = 1)).mean(axis=1)

ax = fig.add_subplot(234)

ax.hist(Derivative,alpha=0.8,bins=50,density=True,range=(-250,250))
ax.hist(Derivative2,alpha=0.8,bins=50,density=True,range=(-250,250))
ax.legend(['No Exoplanets', 'Has Exoplanets'])
ax.set_xlabel('L1 Norm of Derivative')
ax.set_ylabel('Num. of Stars')
```

```
MaxDerivative = np.max(np.gradient(LightCurves[ex_labels==1], axis = 1), axis = 1)
MaxDerivative2 = np.max(np.gradient(LightCurves[ex_labels==2], axis = 1), axis = 1)

ax = fig.add_subplot(235)

ax.hist(MaxDerivative,alpha=0.8,bins=50,density=True,range=(-500,500))
ax.hist(MaxDerivative2,alpha=0.8,bins=50,density=True,range=(-500,500))
ax.legend(['No Exoplanets', 'Has Exoplanets'])
ax.set_xlabel('Max of Derivative')
ax.set_ylabel('Num. of Stars')

MaxSecDerivative = np.max(np.gradient(np.gradient(LightCurves[ex_labels==1], axis = 1), axis = 1), axis␣
 ↪= 1)
MaxSecDerivative2 = np.max(np.gradient(np.gradient(LightCurves[ex_labels==2], axis = 1), axis = 1), axis␣
 ↪= 1)

ax = fig.add_subplot(236)

ax.hist(MaxSecDerivative,alpha=0.8,bins=50,density=True,range=(-500,500))
ax.hist(MaxSecDerivative2,alpha=0.8,bins=50,density=True,range=(-500,500))
ax.legend(['No Exoplanets', 'Has Exoplanets'])
ax.set_xlabel('Max of Second Derivative')
ax.set_ylabel('Num. of Stars')
```

```
In the data set there are: 5050 Stars without exoplanets.
In the data set there are: 37 Stars without exoplanets.
```

```
[4]: Text(0, 0.5, 'Num. of Stars')
```



Unfortunately none of our clever statistics seem to really separate the data. It seems like stars with exoplanets may have higher max derivatives, but this only holds for the distribution and does not make for a simple test yet. We need to actually perform machine learning. Let us use an all purpose weapon, the support vector machine:

[5]:
```
from sklearn import svm
SupportVectorClassifier = svm.SVC()
SupportVectorClassifier.fit(LightCurves, ex_labels);
```

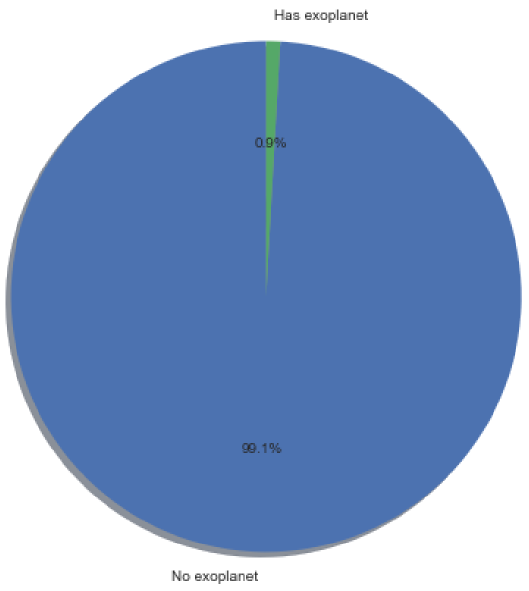
We have trained the support vector machine on the data. Now let us evaluate how well this trained algorithm performs on a test set.

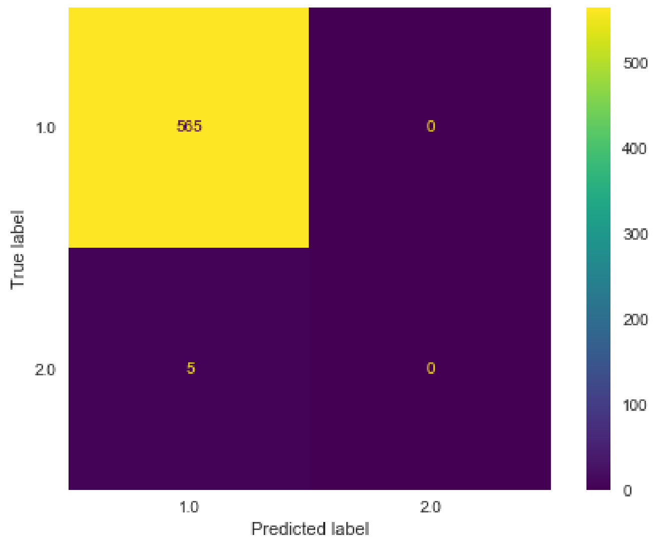
[6]:
```
data_test = pd.read_csv("exoTest.csv")

TestLightCurves = data_test.values[:, 1:]
TestLabels = data_test.values[:, 0]

prediction=SupportVectorClassifier.predict(TestLightCurves)
from sklearn.metrics import accuracy_score, confusion_matrix, plot_confusion_matrix


print('Accuracy Score: {}'
      .format(accuracy_score(TestLabels,prediction)))
```

```
Accuracy Score: 0.9912280701754386
```

On first sight, we have achieved 99.12% accuracy on the test set, which seems nice. But let us dig a little bit deeper by also printing the confusion matrix. Which is a matrix $C = (C_{i,j})_{i,j=1}^2$. Where $C_{0,0}$ denotes the number of true negatives, $C_{1,1}$ are true positives, $C_{1,0}$ are false negatives, $C_{0,1}$ are false positives

[7]:
```
fig = plt.figure(figsize=(8,8))
plt.pie([np.sum(TestLabels==1), np.sum(TestLabels==2)], labels=['No exoplanet', 'Has exoplanet'],␣
 ↪autopct='%1.1f%%',
        shadow=True, startangle=90)
plt.show()



plot_confusion_matrix(SupportVectorClassifier, TestLightCurves, TestLabels)
plt.grid(False)
print('Confusion Matrix:\n {}'
      .format(confusion_matrix(TestLabels,prediction)))
```





```
Confusion Matrix:
 [[565   0]
 [  5   0]]
```

The confusion matrix and the pie chart show quite clearly what is the problem. The data set is very imbalanced. The classifier, while achieving high accuracy, was not successfull in labelling only a single star with an exoplanet correctly. In fact, it labelled all stars as having no exoplanet.

It seems like we have to use a bit more sophisticated methods.

We start by making the data a bit nicer by standardising and filtering it. We filter out high and low frequencies by applying a wavelet transform. We also remove very oscillating element as they seem to be outliers.

Next, we will transform the data so that it is in a format that may exhibit the characteristics that we need to classify. As we have seen in one of the light curves, stars with exoplanets exhibit periodically appearing drops in light intensity. To expose this periodicity, it makes sense to take the Fourier transform. We also want our classifier to be independent of temporal shifts. This can be enforced by taking the absolute value of the Fourier transform, since translation of functions corresponds to modulation of its Fourier transform.

[13]:
```
def filterData(DataSet,wav_len):

    wavelet = gaussian(wav_len, 1)
    wavelet = np.diff(np.diff(wavelet))# Produce a wavelet with two vanishing moments

    for k in range(DataSet.shape[0]):
        DataSet[k,:] = DataSet[k,:] - DataSet[k,:].mean()
        DataSet[k,:] = DataSet[k,:] / DataSet[k,:].std()
        if(np.sum(np.abs(np.diff(DataSet[k,:]))) > 200*max(abs(DataSet[k,:]))):
            DataSet[k,:] = 0;  # remove light curves with too much oscillation
        else:
            DataSet[k,:] = np.convolve(DataSet[k,:], wavelet, 'same')
            DataSet[k,:] = np.abs(fft(DataSet[k,:]))**2

    return DataSet
```

One big problem that we observed was the imbalance of the data set. We attack this problem by generating artificial data. The artificial data is produced ba making signals that have periodic spikes.

[14]:
```
newexoplanets = np.zeros([500, LightCurves.shape[1]])
new_ex_labels = 2*np.ones(500)

for k in range(500):
    newexoplanets[k,:] = LightCurves[500+k,:]
    newexoplanets[k,:] = (newexoplanets[k,:] - newexoplanets[k,:].mean())/newexoplanets[k,:].std()
    period = 100+k
    start = np.random.uniform(3,int(period))
    for j in range(int(start), LightCurves.shape[1]-3, int(period)):
        randpershift = int(np.random.uniform(-1,1))

        newexoplanets[k, j-1  + randpershift] = newexoplanets[k, j-1  + randpershift]-6
        newexoplanets[k,j + randpershift]     = newexoplanets[k,j + randpershift]-12
        newexoplanets[k, j+1 + randpershift]  = newexoplanets[k, j+1+randpershift]-6

LightCurves_Augmented = np.concatenate((LightCurves, newexoplanets))
ex_labels_Augmented = np.concatenate([ex_labels, new_ex_labels])
```

Now we apply the filtering to the augmented data set.

[15]:
```
LightCurves_Augmented_F = filterData(LightCurves_Augmented.copy(), 12)
TestLightCurves_F =filterData(TestLightCurves.copy(), 12)
```

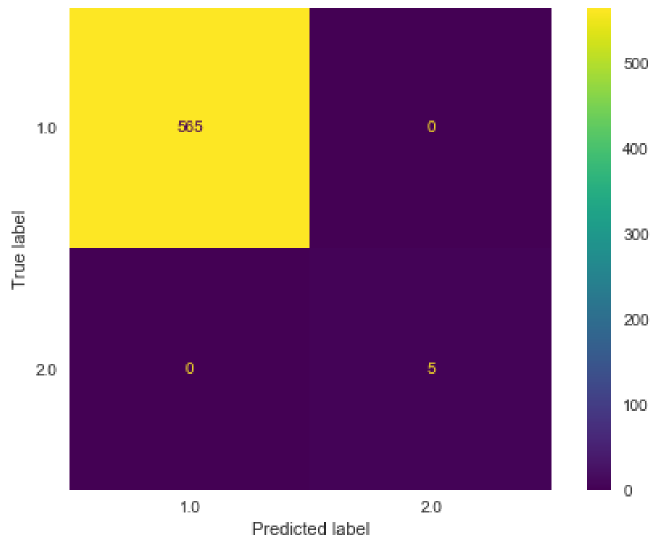
Lets apply our support vector machine again

[16]:
```
SupportVectorClassifier = svm.SVC()
SupportVectorClassifier.fit(LightCurves_Augmented_F, ex_labels_Augmented);
```

…and lets look at the result again:

```
[17]: prediction=SupportVectorClassifier.predict(TestLightCurves_F)

      print('Accuracy Score: {}'
            .format(accuracy_score(TestLabels,prediction)))

      plot_confusion_matrix(SupportVectorClassifier, TestLightCurves_F, TestLabels)
      plt.grid(False)
      print('Confusion Matrix:\n {}'
            .format(confusion_matrix(TestLabels,prediction)))
```

```
Accuracy Score: 1.0
Confusion Matrix:
 [[565   0]
 [  0   5]]
```



... this looks much better. We found all five exoplanets without knowing anything about astrophysics!

1.2 Language of machine learning:

Types of learning:

- *Classification:* Assigning a discrete label to items. Example: Exoplanet yes or no, topics in document classification, or content in image classification.
- *Regression:* Predicting a real value. Example: Prediction of the value of a stock value, temperature or other physical values.
- *Ranking:* Order items according to a criterion. Example: page rank to order webpages according to how well they fit a search query.
- *Clustering:* partitioning of items into subsets. See Figure 1. Example: social networks.
- *Dimensionality reduction/manifold learning:* transform high dimensional data set into a low dimensional representation.

Learning stages:

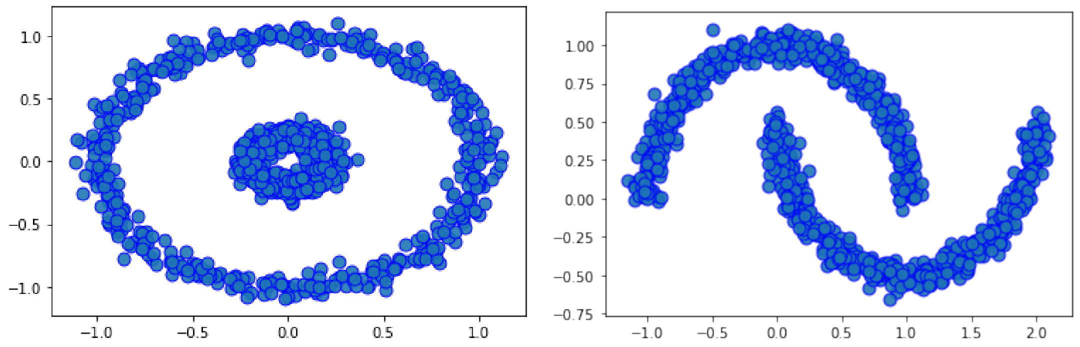


Figure 1: Data sets to cluster

- *Examples:* Observations/Instances of data used in the learning process or to evaluate. Stars in our exoplanet study.
- *Features:* The set of attributes of the examples. In the exoplanet study, these are the light curves.
- *Labels:* Values or categories assigned to the examples. Has exoplanet or does not have exoplanet.
- *Hyperparameters:* Parameters that define the learning algorithm. These are not learned. E.g., number of neurons of the neural networks, when to stop training, e.t.c.
- *Training sample:* These are the examples that are used to train the learning algorithm.
- *Validation sample:* These examples are only indirectly used in the learning algorithm, to tune its hyperparameters.
- *Test sample:* These examples are not accessed during training. After training they are used to determine the accuracy of the algorithm.
- *Loss function:* This function is used to measure the distance between the predicted and true label. If Y is the set of labels, then $L : Y \times Y \to \mathbb{R}^+$. Examples include the zero-one loss: $Y = \{-1, 1\}$, $L_{0-1}(x, y) = \mathbb{1}_{x \neq y}$, the square loss $Y = \mathbb{R}^d$, $L_{sq}(x, y) = \|x - y\|^2$. In the exoplanet study, we used the binary cross entropy: $Y = [0, 1]$, where $L_{ce}(x, y) = -(y \log(x) + (1 - y) \log(1 - x))$ (in our case, the true labels y only take values $\{0, 1\}$).
- *Hypothesis set:* A set of functions that map features to labels.

Learning Scenarios:

- *Supervised learning:* The learner has access to labels for every training and evaluation sample. This was the case in the exoplanet study.
- *Unsupervised learning:* Here we do not have labels. A typical example is clustering.
- *Semi-supervised learning:* Here some of the data have labels. Here the labels and the structure of the data need to be used.
- *Online learning:* Here training and testing are performed iteratively in rounds. In each round we receive new data. We make a prediction receive an evaluation and update our model. The goal is to reduce the so-called regret. This describes how much worse one performed than an expert would in hindsight.
- *Reinforcement learning:* Similar to online learning in the sense that training and testing phases are mixed. The learner receives a reward for each action and seeks to maximise this reward. This if often used to train algorithms to play computer games.

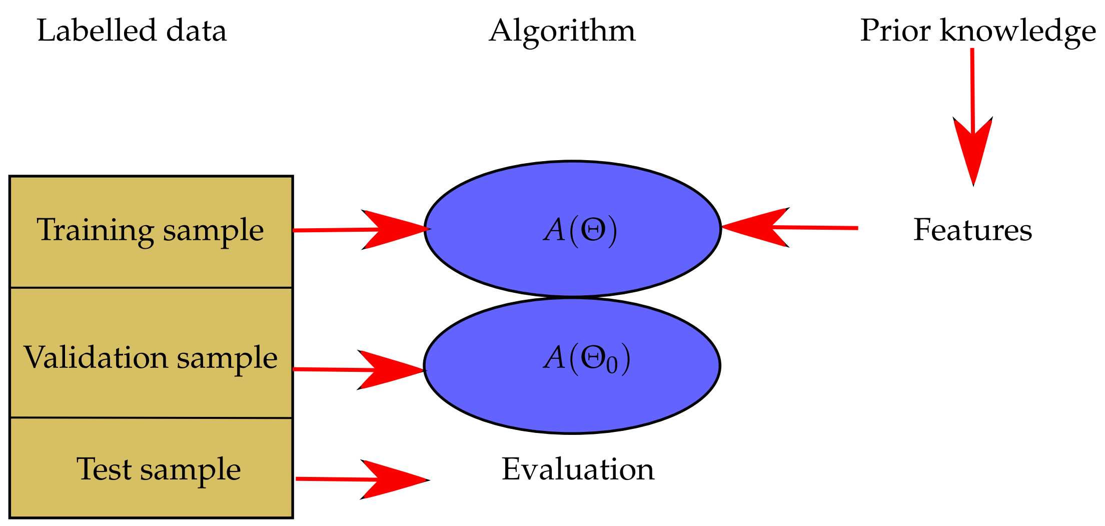


Figure 2: Learning pipeline. We learn using an algorithm $A(\Theta)$. This algorithm can be chosen based on certain features and prior knowledge of the problem. This algorithm has hyperparameters Θ that we can choose based on the validation sample.

- *Active learning:* An oracle exists that can be queried by the learner for labels to samples chosen by the learner.

Generalisation: Generalisation describes the performance of the learned algorithm outside of the training set.

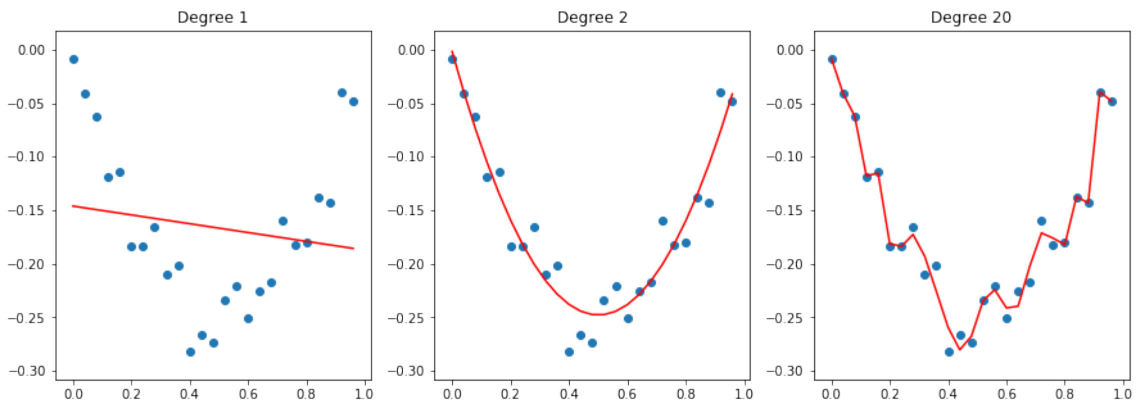
Example:

a) Polynomials fitting points

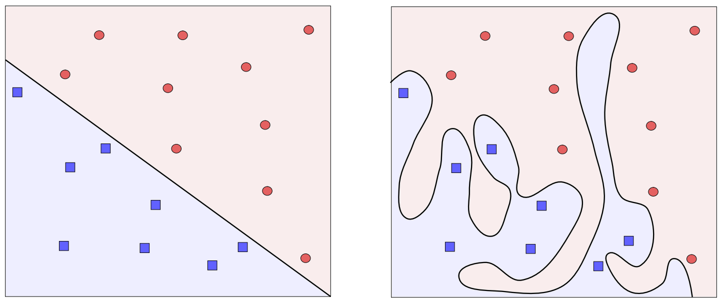
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
N = 25

x = np.arange(0,1, 1/N)

y = x**2 - x + np.random.normal(0, 0.02, N)

polyordLOW = np.poly1d(np.polyfit(x, y, 1))
polyordRIGHT = np.poly1d(np.polyfit(x, y, 2))
polyordHIGH = np.poly1d(np.polyfit(x, y, N-5))

plt.figure(figsize = (15,5))
plt.subplot(1,3,1)
plt.scatter(x,y)
plt.plot(x,polyordLOW(x), c = 'r')
plt.title('Degree 1')
plt.subplot(1,3,2)
plt.scatter(x,y)
plt.title('Degree 2')
plt.plot(x,polyordRIGHT(x), c = 'r')
plt.subplot(1,3,3)
plt.scatter(x,y)
plt.plot(x,polyordHIGH(x), c = 'r')
plt.title('Degree 20')
```

[2]:



b) Binary classification



c) Real world: Sports statistics. "*Red Bull Salzburg never loses a game in the Champions League if they play at home, the moon is full and at least 3 yellow cards are awarded in the first 20 minutes to players with odd jersey numbers.*"

d) Science: Geocentric model. Based on epicycles. See Figure 3.

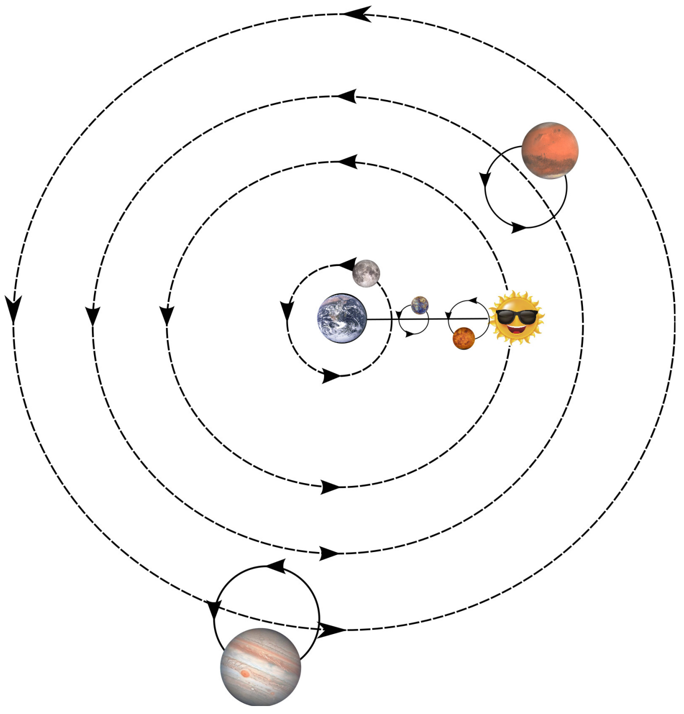


Figure 3: Ptolemaic system

2 Lecture 2 – A Mathematical Framework

2.1 PAC learning framework

- *Input/Example space* $\mathcal{X}$.
- *Output/Label space* $\mathcal{Y}$. (For the rest of this chapter we do *binary classification* $\mathcal{Y} = \{0, 1\}$.)
- *Concept class* $\mathcal{C} \subset \{\mathcal{X} \to \mathcal{Y}\}$. These are possible relationships between examples and labels. We typically assume that we know this. A funtion $c \in \mathcal{C}$ is called a *concept*. There is often one specific concept that we want to identify. We call this the *target concept*. We do not know this.
- *Data distribution* is a distribution $\mathcal{D}$ on $\mathcal{X}$. For simplicity, we assume in the sequel that $\mathcal{D}$ has a density, if $\mathcal{X}$ is not discrete. We do not know this.
- *Hypothesis set* $\mathcal{H} \subset \{\mathcal{X} \to \mathcal{Y}\}$. This does not need to coincide with $\mathcal{C}$.
- Training samples are generated by drawing i.i.d. examples $x_1, \ldots, x_m$ subject to $\mathcal{D}$. The samples are then given as $(x_i, c(x_i))_{i=1}^m$ for a fixed concept c.

Based on the training data, a learning algorithm chooses a function in the hypothesis set. This choice is good, if it is close to an underlying target concept. What is meant by close? We want the generalisation error to be small:

Definition 2.1 (Generalisation error). *Let $h \in \mathcal{H}$, $c \in \mathcal{C}$, and let $\mathcal{D}$ be a data distribution. The* generalisation error or risk *of h is defined as*

$$\mathcal{R}(h) = \mathbb{P}_{x \sim \mathcal{D}} (h(x) \neq c(x)) = \mathbb{E}\left(\mathbb{1}_{h(x) \neq c(x)}\right),$$

where $\mathbb{1}_A$ is the indicator/characteristic function of the event A.[a]

[a] We assume here, that all probabilities are well defined. Of course this restricts the hypothesis and concept classes to some extent. We will ignore all issues of measurability from now on.

In practice, we cannot compute the generalisation error $\mathcal{R}(h)$ since we know neither $\mathcal{D}$ nor the target hypothesis c. We can compute the error on a sample instead:

Definition 2.2 (Empirical error). *Let $h \in \mathcal{H}$, and $S := (x_i, y_i)_{i=1}^m$ be a training sample. The* empirical error or empirical risk *is defined as*

$$\widehat{\mathcal{R}}_S(h) = \frac{1}{m} \sum_{i=1}^m \mathbb{1}_{h(x_i) \neq y_i}.$$

Since the data is generated i.i.d with respect to $\mathcal{D}$, we see that $\widehat{\mathcal{R}}_S(h)$ is an unbiased estimator for $\mathcal{R}(h)$:

$$\begin{aligned}
\mathbb{E}_{(x_i)_{i=1}^m \sim \mathcal{D}^m} (\widehat{\mathcal{R}}_{(x_i, c(x_i))_{i=1}^m})(h) &= \frac{1}{m} \sum_{i=1}^m \mathbb{E}_{(x_i)_{i=1}^m \sim \mathcal{D}^m} (\mathbb{1}_{h(x_i) \neq c(x_i)}) \\
&= \frac{1}{m} \sum_{i=1}^m \mathbb{E}_{x \sim \mathcal{D}} (\mathbb{1}_{h(x) \neq c(x)}) \\
&= \frac{1}{m} \sum_{i=1}^m \mathcal{R}(h) = \mathcal{R}(h).
\end{aligned}$$

We want to learn the target concept from samples. When is this even possible? What does possible even mean?

Definition 2.3 (PAC learnability). *Let $\mathcal{C}$ be a concept class. We say that $\mathcal{C}$ is* PAC-learnable *if there exists a function $m_{\mathcal{C}} : (0,1)^2 \to \mathbb{N}$ and an algorithm $\mathcal{A}$ mapping samples S to functions $\mathcal{A}(S) \in \{\mathcal{X} \to \mathcal{Y}\}$ with the following property: For every distribution $\mathcal{D}$ on $\mathcal{Y}$, for every target concept $c \in \mathcal{C}$, and for all $\epsilon, \delta \in (0,1)$:*

$$\mathbb{P}_{(x_i)_{i=1}^m \sim \mathcal{D}^m} (\mathcal{R}(\mathcal{A}((x_i, c(x_i))_{i=1}^m)) \leq \epsilon) \geq 1 - \delta,$$

if $m \geq m_{\mathcal{C}}(\epsilon, \delta)$.

Note that the definition of PAC learnability is distribution free. Also it describes the worst case behaviour, over the whole concept class.

2.2 An example

- $\mathcal{X} := [0,1]^2$
- $\mathcal{C} := \{\mathbb{1}_{[r_1, r_2] \times [r_3, r_4]} : r_1, r_2, r_3, r_4 \in (0,1)\}$.

Lets define our learning algorithm $\mathcal{A}$ as follows: for $S = (x_i, y_i)_{i=1}^m$ we pick $r_1', r_2', r_3', r_4' \in (0,1)$ so that $[r_1', r_2'] \times [r_3', r_4']$ is the smallest rectangle containing all x_i such that $y_i = 1$ and then $\mathcal{A}(S) = \mathbb{1}_{[r_1', r_2'] \times [r_3', r_4']}$.

Let us analyse the expected error of our algorithm. Pick arbitrary $c \in \mathcal{C}$ and distribution $\mathcal{D}$ on $[0,1]^2$. Let $\epsilon > 0$:

1. Note that for a sample S we have $\{\mathcal{A}(S) = 1\} \subset \{c = 1\}$.

2. The expected error is therefore given by

$$\mathcal{R}(\mathcal{A}(S)) = \mathbb{E}_{\mathcal{D}}(c - \mathcal{A}(S)).$$

3. Assuming, $\mathbb{E}_{\mathcal{D}}(c) > \epsilon$ we choose four rectangles $(R_j)_{j=1}^4$ as in Figure 4 each of probability mass exactly $\epsilon/4$.[1]

4. Observe that, if $\mathbb{E}_{\mathcal{D}}(c - \mathcal{A}(S)) > \epsilon$, then in particular $\mathbb{E}_{\mathcal{D}}(c) > \epsilon$ and $\operatorname{supp}\mathcal{A}(S)$ cannot intersect all 4 rectangles of Step 3. Hence, there is one rectangle that does not contain any training samples. In other words,

$$\begin{aligned}\mathbb{P}_{(x_i)_{i=1}^m \sim \mathcal{D}^m}(\mathcal{R}(\mathcal{A}(S)) > \epsilon) &\leq \sum_{j=1}^{m} \mathbb{P}_{(x_i)_{i=1}^m \sim \mathcal{D}^m}(x_i \notin R_j) \\ &\leq \sum_{j=1}^{4} \prod_{i=1}^{m} \mathcal{P}_{x \sim \mathcal{D}}(x \notin R_j) \\ &\leq 4(1 - (\epsilon/4))^m \leq 4e^{-m\epsilon/4},\end{aligned}$$

where we use the inequality $1 + x \leq e^x$ which holds for all $x \in \mathbb{R}$.[2]

5. Setting $\delta = 4e^{-m\epsilon/4}$ yields that $\mathcal{C}$ is PAC learnable with $m_{\mathcal{C}}(\epsilon, \delta) = (4/\epsilon)\ln(4/\delta)$.

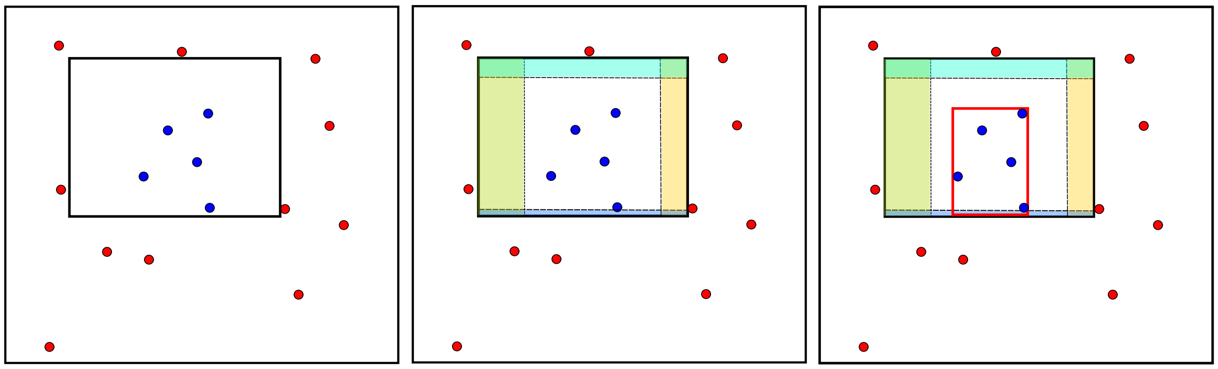


Figure 4: **Left:** A sample, drawn according to $\mathcal{D}$ as well as the target concept. **Middle:** Rectangles of area $\epsilon/4$ each. **Right:** Red box is the solution of $\mathcal{A}$.

2.3 Finite hypothesis, consistent case

We analyse the consistent case now, which is when the concept class $\mathcal{C}$ is a subset of the hypothesis set $\mathcal{H}$ of possible solutions of our learning algorithm.

If $\mathcal{H}$ is finite (and therefore $\mathcal{C}$ is finite) we can get the following learning bound:

[1]This is possible, by adapting the width, because we assumed that $\mathcal{D}$ has a density on the continuous space $\mathcal{X}$.
[2]Note that the argument requires the choice of the $(R_j)_{j=1}^4$ to be independent of $\mathcal{A}$.

Theorem 2.1 (Learning bound, finite $\mathcal{H}$, consistent). *Let $\mathcal{H} \supset \mathcal{C}$ be hypothesis set and concept class. Let $\mathcal{D}$ be a data distribution and $\mathcal{A}$ be an algorithm, such that for each $c \in \mathcal{H}$, and each sample $S = (x_i, c(x_i))_{i=1}^m$ we have that*

$$\widehat{\mathcal{R}}_S(\mathcal{A}(S)) = 0.$$

Then, for every $\delta, \epsilon > 0$, we have that

$$\mathbb{P}_{S \sim D^m}(\mathcal{R}(\mathcal{A}(S)) \leq \epsilon) \geq 1 - \delta,$$

if

$$m \geq \frac{1}{\epsilon}\left(\log|\mathcal{H}| + \log\frac{1}{\delta}\right).$$

In other words, for every $\epsilon, \delta > 0$, with probability at least $1 - \delta$

$$\mathcal{R}(\mathcal{A}(S)) \leq \frac{1}{m}\left(\log|\mathcal{H}| + \log\frac{1}{\delta}\right).$$

Proof.

- Let $\epsilon > 0$ and define $\mathcal{H}_\epsilon = \{h \in \mathcal{H} \colon \mathcal{R}(h) > \epsilon\}$.
- A fixed hypothesis $h \in \mathcal{H}_\epsilon$ fails to match the target concept c on a set Z of measure at least ϵ. If $\widehat{\mathcal{R}}_S(h) = 0$ then this means that we have avoided Z over m random draws subject to $\mathcal{D}$. The probability of this happening is bounded by $(1-\epsilon)^m$.
- We bound the probability that this happens for at least one $h \in \mathcal{H}_\epsilon$ by a union bound:

$$\begin{aligned}\mathbb{P}\left(\exists h \in \mathcal{H}_\epsilon \colon \widehat{\mathcal{R}}_S(h) = 0\right) &\leq \sum_{h \in \mathcal{H}_\epsilon} \mathbb{P}\left(\widehat{\mathcal{R}}_S(h) = 0\right) \\ &\leq \sum_{h \in H_\epsilon} (1-\epsilon)^m \leq |H_\epsilon| e^{-\epsilon m} \leq |H| e^{-\epsilon m}.\end{aligned}$$

- We set $\delta = |H| e^{-\epsilon m}$ and conclude the result.

□

2.4 Finite hypothesis, inconsistent case

If $\mathcal{H} \not\subset \mathcal{C}$, then we can still show that $\mathcal{R}(h)$ is not much larger than $\widehat{\mathcal{R}}_S(h)$ with high probability. We need some preparation first.

Theorem 2.2 (Hoeffding's inequality). *Let $X_1, \ldots, X_m$ be independent random variables such that for all $i \in [m]$[a], $a_i \leq X_i \leq b_i$ almost surely for some $a_i, b_i \in \mathbb{R}$. Then, for $\epsilon > 0$, it holds that with $S_m = \sum_{i=1}^m X_i$*

$$\mathbb{P}(S_m - \mathbb{E}(S_m) > \epsilon) \leq e^{-2\epsilon^2 / \sum_{i=1}^m (b_i - a_i)^2},$$
$$\mathbb{P}(S_m - \mathbb{E}(S_m) < -\epsilon) \leq e^{-2\epsilon^2 / \sum_{i=1}^m (b_i - a_i)^2}.$$

[a] We write $[N] := \{1, \ldots, N\}$

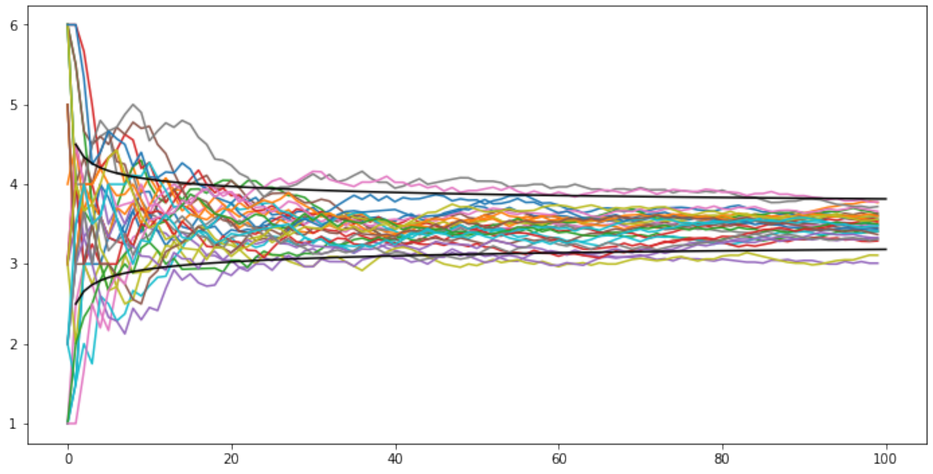
Proof. See [5, Theorem D.2]. □

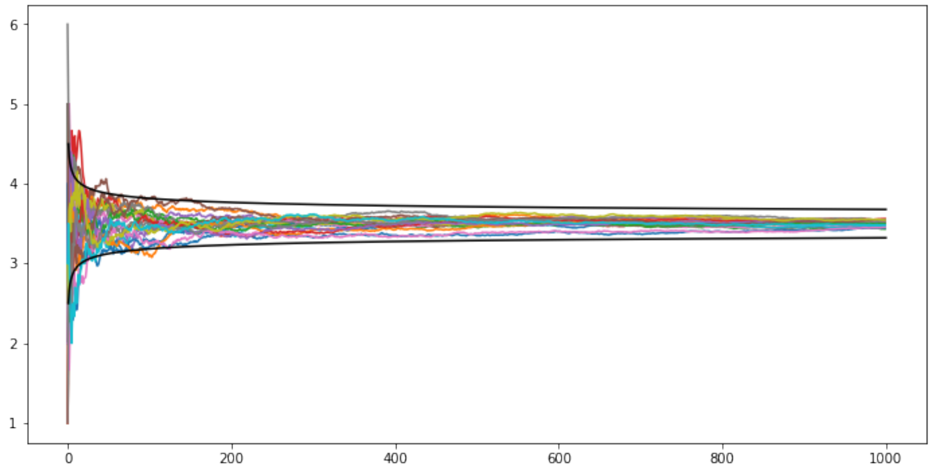
[8]:
```
import numpy as np
import numpy.matlib as mlb
import matplotlib.pyplot as plt
```

Hoeffdings inequality tells us that if we draw a die m times then the mean of the observed eyes should concentrate strongly around 3.5. Indeed since modelling each draw of a die by an iid random variable X_i tking values in $[6]$ yields that

$$\mathbb{P}\left(\left|\frac{1}{m}\sum_{i=1}^{m} X_i - 3.5\right| > m^{-1/4}\right) \leq 2e^{-2\frac{\sqrt{m}}{25}}.$$

[48]:
```
num_of_experiments = 30

for num_of_draws in 100, 1000:
    diceRes = np.random.randint(1,7, [num_of_experiments, num_of_draws])
    scaling = 1/mlb.repmat(np.arange(1,num_of_draws+1), num_of_experiments, 1)
    cum_mean = np.multiply(np.cumsum(diceRes, 1),scaling)

    plt.figure(figsize  = (12,6))
    plt.plot(cum_mean.T)
    plt.plot(np.arange(1,num_of_draws+1), 3.5+np.power(np.arange(1,num_of_draws+1), -1/4), c = 'k')
    plt.plot(np.arange(1,num_of_draws+1), 3.5-np.power(np.arange(1,num_of_draws+1), -1/4), c = 'k')
```





Now we observe the following corollary:

Corollary 2.1. *Let $\epsilon > 0$ and let $\mathcal{D}$ be a distribution on $\mathcal{X}$ and $c : \mathcal{X} \to \{0,1\}$ be a target concept. Then, for every $h : \mathcal{X} \to \{0,1\}$ it holds that*

$$\mathbb{P}_{(x_i)_{i=1}^m \sim \mathcal{D}^m}\left(\widehat{\mathcal{R}}_{(x_i,c(x_i))_{i=1}^m}(h) - \mathcal{R}(h) \geq \epsilon\right) \leq e^{-2m\epsilon^2},$$
$$\mathbb{P}_{(x_i)_{i=1}^m \sim \mathcal{D}^m}\left(\widehat{\mathcal{R}}_{(x_i,c(x_i))_{i=1}^m}(h) - \mathcal{R}(h) \leq -\epsilon\right) \leq e^{-2m\epsilon^2}.$$

In particular

$$\mathbb{P}_{(x_i)_{i=1}^m \sim \mathcal{D}^m}\left(|\widehat{\mathcal{R}}_{(x_i,c(x_i))_{i=1}^m}(h) - \mathcal{R}(h)| \geq \epsilon\right) \leq 2e^{-2m\epsilon^2}.$$

Proof. We have by Definition 2.2 that

- $\mathbb{E}(\widehat{\mathcal{R}}_{(x_i,c(x_i))_{i=1}^m}(h)) = \mathcal{R}(h)$
- $\widehat{\mathcal{R}}_{(x_i,c(x_i))_{i=1}^m}(h) = \sum_{i=1}^m X_i$ for independent random variables X_i with $0 \leq X_i \leq 1/m$ almost surely for $i \in [m]$.

We conclude the proof by applying Theorem 2.2. □

We can extend Corollary 2.1 to any finite hypothesis set by a union bound.

Theorem 2.3 (Learning bound, finite $\mathcal{H}$, inconsistent). *Let $\mathcal{H}$ be a finite hypothesis set. Then, for every $\delta > 0$, the following inequality holds with probability at least $1 - \delta$ over the sample $S = (x_i, c(x_i))_{i=1}^m$:*

$$\mathcal{R}(h) \leq \widehat{\mathcal{R}}_S(h) + \sqrt{\frac{\log|\mathcal{H}| + \log\frac{2}{\delta}}{2m}} \qquad \text{for all } h \in \mathcal{H}. \tag{1}$$

Proof. Let $\mathcal{H} := \{h_1, \dots, h_n\}$. We compute:

$$\begin{aligned} &\mathbb{P}(\exists h_i \in \mathcal{H}\colon |\mathcal{R}(h_i) - \widehat{\mathcal{R}}_S(h_i)| \geq \epsilon) \\ &\leq \sum_{i=1}^n \mathbb{P}(|\mathcal{R}(h_i) - \widehat{\mathcal{R}}_S(h_i)| \geq \epsilon) \\ \text{[Corollary 2.1]} &\leq \sum_{i=1}^n 2e^{-2m\epsilon^2} \leq 2|\mathcal{H}|e^{-2m\epsilon^2}. \end{aligned}$$

Setting $\delta = 2|\mathcal{H}|e^{-2m\epsilon^2}$ and solving for ϵ yields (1). □

Theorem 1 shows an instance of Occam's Razor principle.

3 Lecture 3 – Some Generalisations and Rademacher Complexities

3.1 Agnostic PAC learning:

The notion of *concept class* requires a deterministic relationship between input x drawn according to $\mathcal{D}$ and the label. This is not always sensible. Instead consider a distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$. Below is an example:

[96]:
```
import matplotlib.pyplot as plt
import numpy as np
import joypy as jp
import pandas as pd
data = pd.read_csv("weather_2017.csv")
data.head()
```

[96]:
```
   number  month  day  temp_dailyMin  temp_minGround  temp_dailyMean
0       0      1    1           -6.2            -9.5            -3.7
1       1      1    2           -7.2            -7.5            -1.0
2       2      1    3           -0.7            -3.3             1.1
3       3      1    4            0.9            -0.1             2.9
4       4      1    5           -3.2            -0.8            -0.1
```

Dataset available here: https://www.kaggle.com/zikazika/sickness-and-weather-data?select=weather_2017.csv

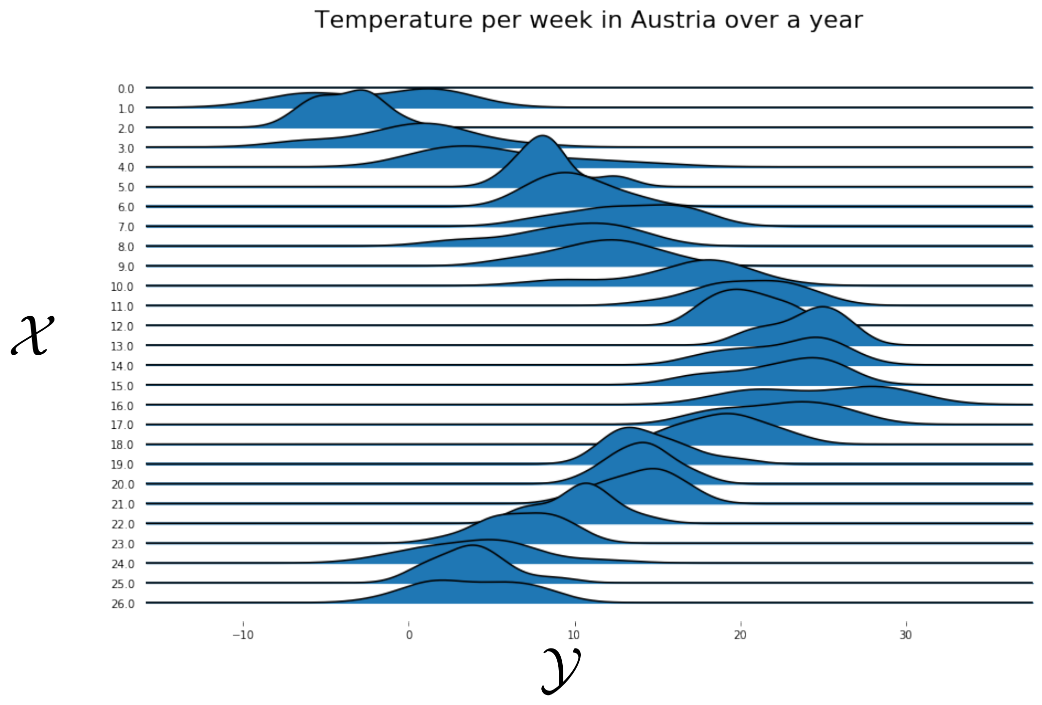
We want to make a plot of temperature vs week. Hence we transform the first column so that each number corresponds to two weeks.

[88]:
```
data["number"] = np.ceil(data ["number"]/14)
```

Below we draw the temperature in Austria over periods of two weeks. We can consider the week number as the example space $\mathcal{X}$ and the temperature as the label space $\mathcal{Y}$.

[93]:
```
# Draw Plot
plt.figure(figsize=(12,8), dpi= 80)
fig, axes = jp.joyplot(data, column='temp_dailyMean', by="number", figsize=(12,8))
plt.title('Temperature per week in Austria over a year', fontsize=22)
plt.show()
```

```
<matplotlib.figure.Figure at 0x7f6639f8e860>
```



If $\mathcal{D}$ is considered as a probability distribution on $\mathcal{X} \times \mathcal{Y}$, then we call the learning problem *stochastic*. Analogously, we call our previous set-up *deterministic*.

In this case, we can redefine the risk to be

$$\mathcal{R}(h) := \mathbb{P}_{(x,y)\sim\mathcal{D}}(h(x) \neq y) = \mathbb{E}_{(x,y)\sim\mathcal{D}}(\mathbb{1}_{h(x)\neq y}). \tag{2}$$

Definition 3.1 (Agnostic PAC learnability). *Let $\mathcal{H}$ a hypothesis set. An algorithm $\mathcal{A}$ mapping samples S to functions in $\mathcal{H}$ is an agnositic PAC learning algorithm if there exists a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ with the following property: for all $\epsilon, \delta \in (0,1)$ and for all distributions $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$*

$$\mathbb{P}_{S\sim\mathcal{D}^m}(\mathcal{R}(\mathcal{A}(S)) - \min_{h\in\mathcal{H}} \mathcal{R}(h)) \leq \epsilon) \geq 1 - \delta,$$

if $m \geq m_{\mathcal{H}}(\epsilon, \delta)$. We call $\mathcal{H}$ agnostic PAC learnable if an agnostic PAC learning algorithm exists.

3.2 Bayes error and noise

In the stochastic case, there does not necessesarily exist any function f such that $\mathcal{R}(f) = 0$.

Definition 3.2 (Bayes error). *Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$. The* Bayes error *$R^*$ is defined as $R^* := \inf_{h\in\mathcal{M}(\mathcal{X},\mathcal{Y})} \mathcal{R}(h)$.*[a]
A hypothesis h such that $\mathcal{R}(h) = R^*$ is called Bayes classifier.

[a]We denote by $\mathcal{M}(\mathcal{X}, \mathcal{Y})$ the set of measurable functions from $\mathcal{X}$ to $\mathcal{Y}$.

We can define a potential Bayes classifier in terms of conditional probabilities:

$$h_{\text{Bayes}}(x) = \arg\max_{y\in\{0,1\}} \mathbb{P}[y|x].$$

For every x we have $\mathbb{P}_{(x,y)\sim\mathcal{D}}(h_{\text{Bayes}}(x) \neq y|x) = \min\{\mathbb{P}_{(x,y)\sim\mathcal{D}}(1|x), \mathbb{P}_{(x,y)\sim\mathcal{D}}(0|x)\}$, which is the smallest possible error. Hence h_{Bayes} is indeed a Bayes classifier.

Definition 3.3 (Noise). *Given a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, we define the* noise *at point $x \in \mathcal{X}$ by* $\text{noise}(x) = \min\{\mathbb{P}_{(x,y)\sim\mathcal{D}}(1|x), \mathbb{P}_{(x,y)\sim\mathcal{D}}(0|x)\}$.
The average noise or simply noise is then defined as $\mathbb{E}(\text{noise}(x))$.

It is clear by construction that

$$\mathbb{E}(\text{noise}(x)) = R^*.$$

The noise level is one aspect describing the hardness of a learning task.

3.3 The Rademacher complexity

We saw that finite hypothesis classes are PAC learnable. Some infinite hypothesis sets seem to be learnable too. This was seen in the example in Section 2.2. We now introduce a new type of complexity that handles infinite hypothesis sets.

Definition 3.4. *Let $a, b \in \mathbb{R}$ and $\mathcal{Z}$ be a set. Let $\mathcal{G} \subset \mathcal{M}(\mathcal{Z}, [a,b])$. Further let $S = (z_1, \dots, z_m) \in \mathcal{Z}^m$. Then the* empirical Rademacher complexity of $\mathcal{G}$ *with respect to S is defined as*

$$\widehat{\mathfrak{R}}_S(\mathcal{G}) = \mathbb{E}_\sigma \left(\sup_{g \in \mathcal{G}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i g(z_i) \right),$$

where $\sigma = (\sigma_1, \dots, \sigma_m)$ with σ_i being i.i.d Rademacher random variables.[a]

[a]This means that the σ_i satisfy $\mathbb{P}(\sigma_i = \pm 1) = 1/2$

Remark 3.1. *The empirical Rademacher complexity measures how well the class $\mathcal{G}$ can correlate with random noise on a given sample S. If, for example G is the set of continuous functions from $[0,1]$ to $[-1,1]$ and S contains m elements $(x_1, \dots, x_m)$ with $x_i \neq x_j$ for all $i, j \in [m]$, then $\widehat{\mathfrak{R}}_S(\mathcal{G}) = 1$. If $\mathcal{G} = \{1\}$ contains only one function then $\widehat{\mathfrak{R}}_S(\mathcal{G}) = 0$.*

The Rademacher complexity is defined for functions with real outputs. To apply it to general learning problems, we introduce the concept of a loss function:

Definition 3.5 (Family of loss functions)**.** *A function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is called a* loss function. *For a hypothesis class $\mathcal{H}$, we define the family of* loss functions associated to $\mathcal{H}$ *by*

$$\mathcal{G} := \{g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \colon g(x,y) = L(h(x), y), h \in \mathcal{H}\}$$

Setting $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ we can apply Definition 3.4 to families of loss functions. We can also define a non-empirical version of the Rademacher complexity.

Definition 3.6. *Let $a, b \in \mathbb{R}$ and $\mathcal{Z}$ be a set. Let $\mathcal{G} \subset \mathcal{M}(\mathcal{Z}, [a,b])$ and let $\mathcal{D}$ be a distribution over $\mathcal{Z}$. For $m \in \mathbb{N}$, we define the* Rademacher complexity *by*

$$\mathfrak{R}_m(\mathcal{G}) := \mathbb{E}_{S \sim \mathcal{D}^m} \left(\widehat{\mathfrak{R}}_S(\mathcal{G}) \right).$$

3.4 Generalisation bound with Rademacher complexity

Below, we present a generalisation bound similar to Theorem 2.3, but for potentially infinite hypothesis sets.

Theorem 3.1. *Let $\mathcal{G} \subset \mathcal{M}(\mathcal{Z}, [0,1])$ and let $\mathcal{D}$ be a distribution on $\mathcal{Z}$. For every $\delta > 0$ and $m \in \mathbb{N}$ we have that for a sample $S = (z_1, \dots, z_m) \sim \mathcal{D}^m$ for all $g \in \mathcal{G}$:*

$$\mathbb{E}(g) \leq \frac{1}{m} \sum_{i=1}^{m} g(z_i) + 2\mathfrak{R}_m(\mathcal{G}) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}} \tag{3}$$

$$\mathbb{E}(g) \leq \frac{1}{m} \sum_{i=1}^{m} g(z_i) + 2\widehat{\mathfrak{R}}_S(\mathcal{G}) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}}, \tag{4}$$

with probability at least $1 - \delta$.

Before we prove this result, lets look at an example:

Let us look at four hypothesis sets: polynomials of degree 3 , 4, 7, and 20. The target concept is a polynomial p_{true} of degree 5. Hence the data distribution constructed by a uniform distribution on $(-1,1)$ and p_{true}. Below, vary the number of sample points, compute the empirical Rademacher complexities of the model with loss function $L(h(x), y) = h(x) - p_{true}(x)$. Note that

$$\mathbb{E}_\sigma \left(\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i (h(x_i) - p_{true}(x_i)) \right) = \mathbb{E}_\sigma \left(\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i h(x_i) \right)$$

We compute the empirical error as

$$\frac{1}{m} \sum_{i=1}^{m} |L(h(x_i), y)|$$

and approximate the expected error $\mathbb{E}(|L(h(x), y)|)$. Note that due to the absolute value, we are not completely in the setup of Theorem 3.1. We will later see, that this does not matter, so we should not overthink this now.

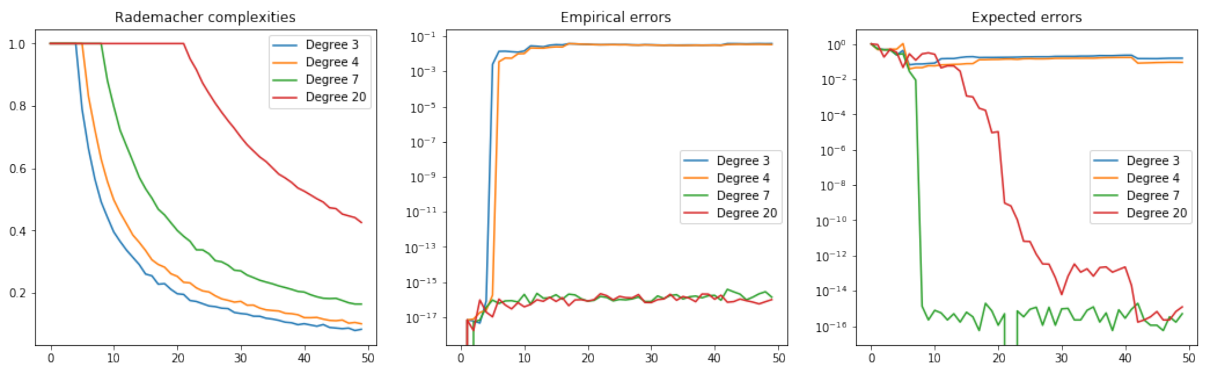
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter('ignore', np.RankWarning)
```

[4]:
```
# set-up
iterations = 50
degrees = [3,4,7,20]
largeNumber = 1000

#errors to be computed
RademacherPoly = np.ones([iterations, len(degrees)])
EmpErrorsPoly = np.zeros([iterations, len(degrees)])
ErrorsPoly = np.ones([iterations, len(degrees)])

#the test data
x_test = np.arange(-1,1,1/largeNumber)
y_test = (x_test - 0.3)* (x_test + 0.15) * x_test * (x_test + 0.75)  * (x_test - 0.8)

# precompute training data on random points:
x = np.random.uniform(-1,1, iterations)
y = (x - 0.3) * (x + 0.15) * x * (x + 0.75)  * (x - 0.8)
```

```
for m in range(1,iterations):

    # take subset of length m from training data
    x_short = x[0:m]
    y_short = y[0:m]

    for k in range(len(degrees)):
        # fit polynomials to data:
        p = np.poly1d(np.polyfit(x_short, y_short, degrees[k]))

        #compute errors
        y_exp = p(x_test) - y_test
        y_emp = p(x_short) - y_short
        EmpErrorsPoly[m, k] = abs(y_emp).mean()
        ErrorsPoly[m, k] = abs(y_exp).mean()

        #estimate empirical Rademacher complexities:
        err = 0
        for it in range(largeNumber):
            rdm = 2*np.round(np.random.uniform(0,1, m))-1
            p = np.poly1d(np.polyfit(x_short, rdm, degrees[k]))
            err = err + np.dot(p(x_short), rdm)/m

        RademacherPoly[m, k] = err/largeNumber
```

[5]:
```
plt.figure(figsize = (18,5))

plt.subplot(131)
plt.plot(np.arange(iterations), RademacherPoly)
plt.legend(('Degree 3', 'Degree 4', 'Degree 7', 'Degree 20'))
plt.title('Rademacher complexities')



plt.subplot(132)
plt.semilogy(np.arange(iterations), EmpErrorsPoly)
plt.legend(('Degree 3', 'Degree 4', 'Degree 7', 'Degree 20'))
plt.title('Empirical errors')

plt.subplot(133)
plt.semilogy(np.arange(iterations), ErrorsPoly)
plt.legend(('Degree 3', 'Degree 4', 'Degree 7', 'Degree 20'))
plt.title('Expected errors')
```

[5]:



Having understood the nature of Theorem 3.1, we can now look its proof. We need the following result:

Theorem 3.2 (McDiarmid's inequality). *Let $m \in \mathbb{N}$, and $X_1, \ldots, X_m$ be independent random variables taking values in $\mathcal{X}$. Assume that there exist $c_1, \ldots, c_m > 0$ and a function $f : \mathcal{X}^m \to \mathbb{R}$ satisfying*

$$|f(x_1, \ldots, x_i, \ldots, x_m) - f(x_1, \ldots, x_i', \ldots, x_m)| \leq c_i,$$

for all $i \in [m]$ and all points $x_1, \ldots, x_m, x_i' \in \mathcal{X}$. Then the following inequalities hold for all $\epsilon > 0$:

$$\mathbb{P}\left(f(S) - \mathbb{E}(f(S)) \geq \epsilon\right) \leq e^{-2\epsilon^2/|c|^2}$$
$$\mathbb{P}\left(f(S) - \mathbb{E}(f(S)) \leq -\epsilon\right) \leq e^{-2\epsilon^2/|c|^2},$$

where $S = (X_1, \ldots, X_m)$ and $c = (c_1, \ldots, c_m)$.

Proof of Theorem 3.1. We define two short-hand notations for a sample $S = (z_1, \ldots, z_m)$:

$$\widehat{\mathbb{E}}_S(g) := \frac{1}{m} \sum_{i=1}^{m} g(z_i)$$
$$\Phi(S) := \sup_{g \in \mathcal{G}} \left(\mathbb{E}(g) - \widehat{\mathbb{E}}_S(g)\right)$$

To prove the theorem, we need to bound $\Phi(S)$ and we will use McDiarmids inequality for this.

Let S and S' be two samples that differ in exactly one point, i.e., $S = (z_1, \ldots, z_i, \ldots, z_m)$ and $S = (z_1, \ldots, z_i', \ldots, z_m)$. We compute:

$$\begin{aligned}
\Phi(S') - \Phi(S) &= \sup_{g \in \mathcal{G}} \left(\mathbb{E}(g) - \widehat{\mathbb{E}}_{S'}(g)\right) - \sup_{g \in \mathcal{G}} \left(\mathbb{E}(g) - \widehat{\mathbb{E}}_S(g)\right) \\
&\leq \sup_{g \in \mathcal{G}} \left(\widehat{\mathbb{E}}_S(g) - \widehat{\mathbb{E}}_{S'}(g)\right) \\
&\leq \sup_{g \in \mathcal{G}} \frac{g(z_i) - g(z_i')}{m} \leq \frac{1}{m},
\end{aligned}$$

where the first inequality, is due to elementary properties of suprema, the second follows from the definition of $\widehat{\mathbb{E}}_S(g)$ and S, S' and the last is due to the fact that g takes values in $[0, 1]$.

The choice of S, S' was arbitrary, and so we conclude that

$$|\Phi(S') - \Phi(S)| \leq \frac{1}{m}$$

for all S, S' differing in one point only. By McDiarmids inequality, we have that for a random sample S

$$\mathbb{P}\left(\Phi(S) - \mathbb{E}_S(\Phi(S)) \geq \epsilon\right) \leq e^{-2\epsilon^2 m}.$$

Hence with probability $1 - \delta/2$

$$\Phi(S) \leq \mathbb{E}_S(\Phi(S)) + \sqrt{\frac{\log(\frac{2}{\delta})}{2m}}. \tag{5}$$

Let us compute $\mathbb{E}_S(\Phi(S))$ next.

$$\mathbb{E}_S(\Phi(S)) = \mathbb{E}_S\left(\sup_{g\in\mathcal{G}}\left(\mathbb{E}(g) - \widehat{\mathbb{E}}_S(g)\right)\right) = \mathbb{E}_S \sup_{g\in\mathcal{G}} \mathbb{E}_{S'}\left(\widehat{\mathbb{E}}_{S'}(g) - \widehat{\mathbb{E}}_S(g)\right), \tag{6}$$

where S' is a sample that is independent from and distributed like S. We used that $\mathbb{E}_{S'}(\widehat{\mathbb{E}}_{S'}(g)) = \mathbb{E}(g)$. By the monotonicity of the expected value, we have that

$$\begin{aligned}\mathbb{E}_S \sup_{g\in\mathcal{G}} \mathbb{E}_{S'}\left(\widehat{\mathbb{E}}_{S'}(g) - \widehat{\mathbb{E}}_S(g)\right) &\leq \mathbb{E}_{S,S'} \sup_{g\in\mathcal{G}}\left(\widehat{\mathbb{E}}_{S'}(g) - \widehat{\mathbb{E}}_S(g)\right)\\ &= \mathbb{E}_{S,S'} \sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^m g(z_i') - g(z_i).\end{aligned}$$

Assume next, that σ is a Rademacher random variable. Then it holds that

$$\mathbb{E}_{S,S'} \sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^m g(z_i') - g(z_i) = \mathbb{E}_\sigma \mathbb{E}_{S,S'} \sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^m \sigma_i(g(z_i') - g(z_i)). \tag{7}$$

To see why this holds, we observe that for every fixed σ a negative sign of σ_i corresponds to switching z_i and z_i' in $\sum_{i=1}^m g(z_i') - g(z_i)$. Since we all z_i, z_i' are chosen i.i.d and we are taking the expectation, this does not affect the output. Applying the sub-additivity of the supremum to (7) yields that

$$\begin{aligned}\mathbb{E}_\sigma \mathbb{E}_{S,S'} \sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^m \sigma_i(g(z_i') - g(z_i)) &\leq \mathbb{E}_\sigma \mathbb{E}_{S'} \sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^m \sigma_i g(z_i') + \mathbb{E}_\sigma \mathbb{E}_S \sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^m -\sigma_i g(z_i)\\ &\leq 2\mathbb{E}_\sigma \mathbb{E}_S \sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^m \sigma_i g(z_i) = 2\mathfrak{R}_m(\mathcal{G}),\end{aligned}$$

where the last inequality follows since σ and $-\sigma$ have the same distribution. This yields (3).

To prove (4), we apply McDiarmids inequality again. Note that for two samples S, S' differing in one point only

$$\widehat{\mathfrak{R}}_S(\mathcal{G}) - \widehat{\mathfrak{R}}_{S'}(\mathcal{G}) \leq \frac{1}{m}$$

and hence with probability $1 - \delta/2$

$$\mathfrak{R}_m(\mathcal{G}) = \mathbb{E}(\widehat{\mathfrak{R}}_S(\mathcal{G})) \leq \widehat{\mathfrak{R}}_S(\mathcal{G}) + \sqrt{\frac{\log\frac{2}{\delta}}{2m}}. \tag{8}$$

Therefore, we conclude with a union bound from (8) and (5) that with probability $1 - \delta$

$$\Phi(S) \leq 2\widehat{\mathfrak{R}}_S(\mathcal{G}) + 3\sqrt{\frac{\log(\frac{2}{\delta})}{2m}}$$

which yields (4). □

4 Lecture 4 – Application of Rademacher Complexities and Growth Function

4.1 Rademacher complexity bounds for binary classification

Theorem 3.1 holds for general families of loss functions. We want to make this notion more concrete for common learning problems.

Lemma 4.1. *Let $\mathcal{H} \subset \mathcal{M}(\mathcal{X}, \{-1,1\})$. Furthermore, let $\mathcal{G} = \{\mathcal{X} \times \mathcal{Y} \ni (x,y) \mapsto \mathbb{1}_{h(x)\neq y} \colon h \in \mathcal{H}\}$. For a sample $(x_i, y_i)_{i=1}^m = S \in (\mathcal{X} \times \mathcal{Y})^m$ we denote $S_{\mathcal{X}} = (x_i)_{i=1}^m$. It holds that*

$$\widehat{\mathfrak{R}}_S(\mathcal{G}) = \frac{1}{2}\widehat{\mathfrak{R}}_{S_{\mathcal{X}}}(\mathcal{H}).$$

Proof. The proof follows from a simple computation which is fundamentally based on the identity: $\mathbb{1}_{h(x)\neq y} = (1 - h(x)y)/2$. With this, we have that

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(\mathcal{G}) &= \mathbb{E}_\sigma\left(\sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^m \sigma_i g(z_i)\right)\\
&= \mathbb{E}_\sigma\left(\sup_{h\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^m \sigma_i \mathbb{1}_{h(x_i)\neq y_i}\right)\\
&= \mathbb{E}_\sigma\left(\sup_{h\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^m \sigma_i \frac{1-h(x_i)y_i}{2}\right)\\
&= \frac{1}{2}\mathbb{E}_\sigma\left(\sup_{h\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^m (-\sigma_i y_i)h(x_i)\right) = \frac{1}{2}\widehat{\mathfrak{R}}_{S_{\mathcal{X}}}(\mathcal{H}),
\end{aligned}$$

where the last identity follows since $(-\sigma_i y_i)$ and σ_i have the same distribution. □

Now we can transfer our generalisation bound of Theorem 3.1 to the binary classification setting:

Theorem 4.1. *Let $\mathcal{H} \subset \mathcal{M}(\mathcal{X}, \{-1,1\})$ and $\mathcal{D}$ be a distribution on $\mathcal{X}$. Then, for every $\delta > 0$ it holds with probability at least $1-\delta$ that*

$$\mathcal{R}(h) \leq \widehat{\mathcal{R}}_S(h) + \mathfrak{R}_m(\mathcal{H}) + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}$$

$$\mathcal{R}(h) \leq \widehat{\mathcal{R}}_S(h) + \widehat{\mathfrak{R}}_S(\mathcal{H}) + 3\sqrt{\frac{\log\frac{2}{\delta}}{2m}},$$

where $S \sim \mathcal{D}^m$.

For the binary loss computing the empirical Rademacher complexity of a hypothesis class $\mathcal{H}$ amounts to solving for all choices of a Rademacher vector an optimisation problem over the whole class $\mathcal{H}$. This can be computationally challenging, if $\mathcal{H}$ is very complex and m is large. Moreover, computing $\mathfrak{R}_m$ is often not possible at all, since we do not know the underlying distribution.

4.2 The growth function

Definition 4.1 (Growth function)**.** *For a hypothesis set $\mathcal{H} \subset \{h \colon \mathcal{X} \to \{-1,1\}\}$, the* growth function *$\Pi_{\mathcal{H}} \colon \mathbb{N} \to \mathbb{N}$ is defined by*

$$\Pi_{\mathcal{H}}(m) = \max_{\{x_1,\dots,x_m\}\subset\mathcal{X}} \left|\{(h(x_1),\dots,h(x_m)) \colon h \in \mathcal{H}\}\right|.$$

The growth function describes the number of ways m points could be grouped into two classes by elements in $\mathcal{H}$. The growth function is independent of the underlying distribution and a useful tool to bound the Rademacher complexity.

A helpful result here is Massart's lemma:

Theorem 4.2 (Massart's Lemma)**.** *Let $A \subset \{x = (x_1, \dots, x_m) \in \mathbb{R}^m : |x| \leq r\}$ be finite set. Then*

$$\mathbb{E}_\sigma \left[\frac{1}{m} \sup_{x \in A} \sum_{i=1}^m \sigma_i x_i \right] \leq \frac{r\sqrt{2 \log |A|}}{m},$$

where the σ_i are independent Rademacher random variables.

Now we can show the following upper bound on the Rademacher complexity:

Corollary 4.1. *Let $\mathcal{H} \subset \{h \colon \mathcal{X} \to \{-1, 1\}\}$. Let $\mathcal{D}$ be a distribution on $\mathcal{X}$. Then, for every $m \in \mathbb{N}$ it holds that,*

$$\mathfrak{R}_m(\mathcal{H}) \leq \sqrt{\frac{2 \log \Pi_{\mathcal{H}}(m)}{m}}.$$

Proof. Notice that every vector of length m with entries either plus or minus one has euclidean norm $\sqrt{m}$. Hence we have that for every sample $S = (x_1, \dots, x_m)$ the set

$$\mathcal{H}_S := \{h(S) \colon h \in \mathcal{H}\}$$

is contained in the $\sqrt{m}$ ball and per definition $|\mathcal{H}_S| \leq \Pi_{\mathcal{H}}(m)$.

Therefore

$$\mathfrak{R}_m(\mathcal{H}) = \mathbb{E}_S \mathbb{E}_\sigma \left(\frac{1}{m} \sup_{h \in \mathcal{H}} \sum_{i=1}^m \sigma_i h(x_i) \right) = \mathbb{E}_S \mathbb{E}_\sigma \left(\frac{1}{m} \sup_{u \in \mathcal{H}_S} \sum_{i=1}^m \sigma_i u_i \right) \leq \sqrt{\frac{2 \log \Pi_{\mathcal{H}}(m)}{m}},$$

by Theorem 4.2. □

Using this estimate, we can reformulate our previous generalisation bound that was formulated in terms of Rademacher complexity via the growth function instead:

Corollary 4.2. *Let $\mathcal{H} \subset \{h \colon \mathcal{X} \to \{-1, 1\}\}$. Then, for any $\delta > 0$, with probability at least $1 - \delta$ for any $h \in \mathcal{H}$:*

$$\mathcal{R}(h) \leq \widehat{\mathcal{R}}_S(h) + \sqrt{\frac{2 \log \Pi_{\mathcal{H}}(m)}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}.$$

4.3 The Vapnik–Chevronenkis Dimension

Definition 4.2 (Shattering)**.** *For a function $h \colon \mathcal{X} \mapsto \{-1, 1\}$, we denote for a set of points $S = (x_1, \dots, x_m) \in \mathcal{X}^m$ by h_S the restriction of h to S. For a hypothesis class $\mathcal{H} \subset \{h \colon \mathcal{X} \to \{-1, 1\}\}$, we say that S is* shattered by *$\mathcal{H}$, if $|\{h_S \colon h \in \mathcal{H}\}| = 2^m$.*

The VC-dimension of a hypothesis class is now the size of the largest set, that is shattered by a hypothesis class. We can equivalently state it in terms of the growth function:

Definition 4.3 (VC-Dimension)**.** *Let $\mathcal{H} \subset \{h \colon \mathcal{X} \to \{-1, 1\}\}$. Then we define the* VC-Dimension *of $\mathcal{H}$ by*

$$\text{VCdim}(\mathcal{H}) = \max \{m \in \mathbb{N} \colon \Pi_{\mathcal{H}}(m) = 2^m\}.$$

Example 4.1 (Intervals)**.** *Let $\mathcal{H} = \{2\mathbb{1}_{[a,b]} - 1 \colon a, b \in \mathbb{R}\}$.*

It is clear that $\mathrm{VCdim}(\mathcal{H}) \geq 2$ *since for $x_1 < x_2$ the functions*

$$2\mathbb{1}_{[x_1-2,x_1-1]} - 1, \quad 2\mathbb{1}_{[x_1-2,x_1]} - 1, \quad 2\mathbb{1}_{[x_1,x_2]} - 1, \quad 2\mathbb{1}_{[x_2,x_2+1]} - 1,$$

are all different, when restricted to $S = (x_1, x_2)$.

On the other hand, if $x_1 < x_2 < x_3$ then, we have that since $h^{-1}(\{1\})$ is an interval for all $h \in \mathcal{H}$ that $h(x_1) = 1 = h(x_3)$ implies $h(x_2) = 1$. Hence, no set of three elements can be shattered. Therefore, $\mathrm{VCdim}(\mathcal{H}) = 2$*. The situation is depicted in Figure 5.*

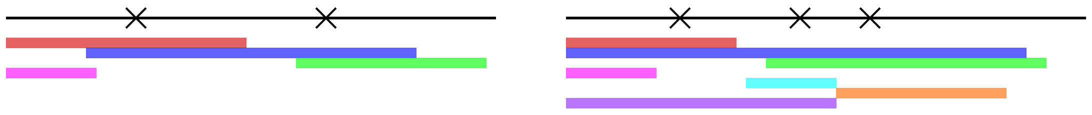


Figure 5: Different ways to classify two or three points. The coloured-blocks correspond to the intervals $[a, b]$.

Example 4.2 (Two dimensional half-spaces)**.** *Let $\mathcal{H} = \{2\mathbb{1}_{\mathbb{R}^+}(\langle a, \cdot \rangle + b) - 1 \colon a \in \mathbb{R}^2, b \in \mathbb{R}\}$ be a hypothesis set of rotated and shifted two-dimensional half-spaces. By Figure 6, we see that $\mathcal{H}$ shatters a set of three points.*

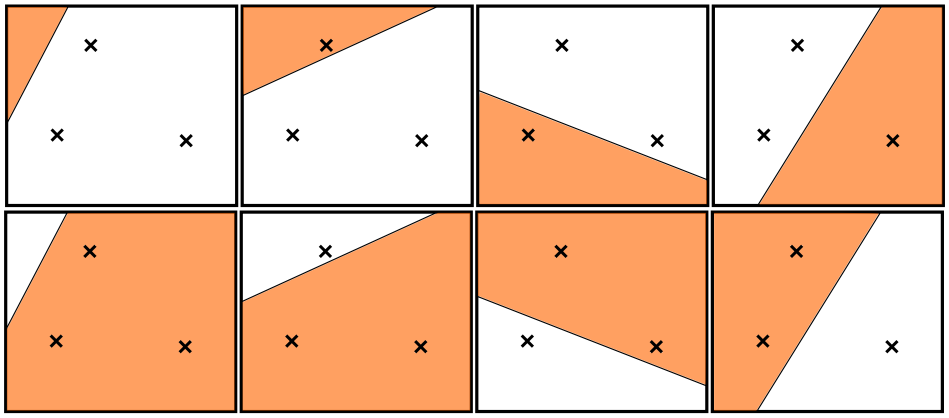


Figure 6: Different ways to classify three by a half-space.

For any four points (x_1, x_2, x_3, x_4) one of two situations will happen. Either one point is in the convex hull of the remaining three or the four points form the edges of a convex quadrilateral. In the first case, we can assume that without loss of generality x_4 is a convex combination of x_1, x_2, x_3. Since half-spaces are convex too, we have that if $h(x_1) = h(x_2) = h(x_3) = 1$ then $h(x_4) = 1$. Therefore, we cannot shatter sets of this form. If, on the other hand, the points (x_1, x_2, x_3, x_4) are the sides of a convex quadrilateral, then, without loss of generality the points x_1 and x_3 lie on different sides of the line connecting x_2 and x_4. Since (x_1, x_2, x_3, x_4) are the extreme points of the quadrilateral, it must be the case that the lines connecting x_1 and x_3 and x_2 and x_4 intersect. Further, any half-space that contains x_1 and x_3 contains by convexity also the line between x_1 and x_3. Any half-space not containing x_2 and x_4 contains, by convexity also no element of the line between x_2, x_4. Hence, there is no half space containing x_1, x_3 but not x_2 and x_4. A visualisation of the argument above is given in Figure 7.

We conclude that for the half-space classifier $\mathrm{VCdim}(\mathcal{H}) = 3$.

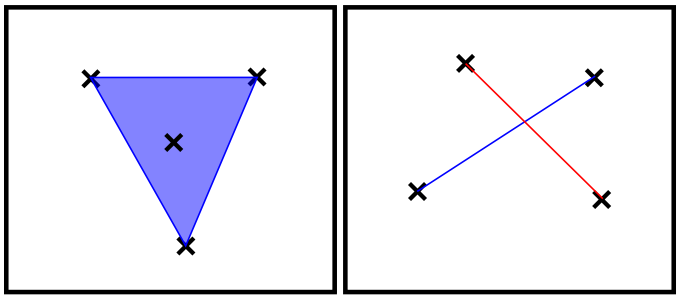


Figure 7: Visualisation of the argument prohibiting shattering of sets of four elements.

The half-space VC dimension bound generalises to arbitrary dimensions.

Example 4.3 (Half-spaces)**.** *Let $d \in \mathbb{N}$, $\mathcal{H} = \{2\mathbb{1}_{\mathbb{R}^+}(\langle a, \cdot \rangle + b) - 1 \colon a \in \mathbb{R}^d, b \in \mathbb{R}\}$ be a hypothesis set of rotated and shifted half spaces. Then,* $\mathrm{VCdim}(\mathcal{H}) = d + 1$.

4.4 Generalisation bounds via VC-dimension

First we are looking for connections between the VC dimension and the growth function.

Theorem 4.3. *Let $\mathcal{H} \subset \{h \colon \mathcal{X} \to \{-1, 1\}\}$ be such that* $\mathrm{VCdim}(\mathcal{H}) = d$*. Then for all $m \in \mathbb{N}$:*

$$\Pi_{\mathcal{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i}. \tag{9}$$

In particular, for all $m \geq d$

$$\log \Pi_{\mathcal{H}}(m) \leq d \log\left(\frac{em}{d}\right) = \mathcal{O}(d \log(m)).$$

Proof. We prove this by induction over $m + d \leq k$. For $k = 2$, we have the options $m = 1$ and $d = 0, 1$ as well as $m = 2, d = 0$.

1. If $d = 0$ and $m \in \mathbb{N}$, then $|\mathcal{H}_S| \leq 1$ for all samples S of size 1 and hence $\Pi_{\mathcal{H}}(1) \leq 1$. Moreover, if for an $m \in \mathbb{N}$, $\Pi_H(m) > 1$, then there would exist a set S with m samples on which $|\mathcal{H}_S| > 1$. That means that on at least one of the elements of S, $\mathcal{H}_S$ takes at least two different values and hence $\Pi_H(1) > 1$, a contradiction. Hence $\Pi_H(m) \leq 1$ for all $m \in \mathbb{N}$. The right-hand side of (9) is always at least 1.

2. If $d \geq 1$ and $m = 1$, then $\Pi_H(1) \leq 2$ per definition, which is always bounded by the right-hand side of (9).

Assume now that the statement (9) holds for all $m + d \leq k$ and let $\bar{m} + \bar{d} = k + 1$. By Points 1 and 2 above, we can assume without loss of generality that $\bar{m} > 1$ and $\bar{d} > 0$.

Let $S = \{x_1, \ldots, x_{\bar{m}}\}$ be a set so that $\Pi_{\mathcal{H}}(\bar{m}) = |\mathcal{H}_S|$ and let $S' = \{x_1, \ldots, x_{\bar{m}-1}\}$.

Let us define an auxiliary set

$$\mathcal{G} := \{h \in \mathcal{H}_{S'} \colon \exists h', h'' \in \mathcal{H}_S, h'(x_{\bar{m}}) \neq h''(x_{\bar{m}}), h = h'_{S'} = h''_{S'}\}. \tag{10}$$

In words, $\mathcal{G}$ contains all those maps in $\mathcal{H}_{S'}$ that have two corresponding functions in $\mathcal{H}_S$.

Now it is clear that

$$|\mathcal{H}_S| = |\mathcal{H}_{S'}| + |\mathcal{G}|. \tag{11}$$

Per assumption $(\bar{m}-1)+\bar{d} \leq k$ and $(\bar{m}-1) \in \mathbb{N}$. Hence, by the induction hypothesis:

$$|\mathcal{H}_{S'}| \leq \Pi_{\mathcal{H}}(\bar{m}-1) \leq \sum_{i=0}^{\bar{d}} \binom{\bar{m}-1}{i}. \tag{12}$$

Note that $\mathcal{G}$ is a set of functions defined on S'. Hence we can compute its VC dimension. If a set $Z \subset S'$ is shattered by $\mathcal{G}$, then $Z \cup \{x_{\bar{m}}\}$ is shattered by $\mathcal{H}_S$. We conclude that

$$\mathrm{VCdim}(\mathcal{G}) \leq \mathrm{VCdim}(\mathcal{H}_S) - 1 \leq \mathrm{VCdim}(\mathcal{H}) - 1 = \bar{d} - 1.$$

Since, by assumption $\bar{d}-1 \geq 0$, we conclude with the induction hypothesis, that

$$|\mathcal{G}| \leq \Pi_{\mathcal{G}}(\bar{m}-1) \leq \sum_{i=0}^{\bar{d}-1} \binom{\bar{m}-1}{i}. \tag{13}$$

We conclude with (11), (12), and (13) that

$$\Pi_{\mathcal{H}}(\bar{m}) = |\mathcal{H}_S| = |\mathcal{H}_{S'}| + |\mathcal{G}| \leq \sum_{i=0}^{\bar{d}} \binom{\bar{m}-1}{i} + \sum_{i=0}^{\bar{d}-1} \binom{\bar{m}-1}{i} = \sum_{i=0}^{\bar{d}} \binom{\bar{m}}{i}.$$

This completes the induction step and yields (9).

Now let us address the 'in particular' part:

We have for $m > d$ by (9) that

$$\begin{aligned}
\Pi_{\mathcal{H}}(m) &\leq \sum_{i=0}^{d} \binom{m}{i} \\
&\leq \sum_{i=0}^{d} \binom{m}{i} \left(\frac{m}{d}\right)^{d-i} \\
&\leq \sum_{i=0}^{m} \binom{m}{i} \left(\frac{m}{d}\right)^{d-i} \\
&= \left(\frac{m}{d}\right)^{d} \sum_{i=0}^{m} \binom{m}{i} \left(\frac{d}{m}\right)^{i}.
\end{aligned}$$

The binomial theorem states that

$$\sum_{i=0}^{m} \binom{m}{i} x^{m-i} y^i = (x+y)^m.$$

In particular, setting $x = 1$ and $y = d/m$, we conclude that

$$\Pi_{\mathcal{H}}(m) \leq \left(\frac{m}{d}\right)^{d} \left(1 + \frac{d}{m}\right)^{m} \leq \left(\frac{m}{d}\right)^{d} e^{d}. \tag{14}$$

The result follows by applying the logarithm to (14). □

Plugging Theorem 4.3 into Corollary 4.2, we can now state a generalisation bound for binary classification in terms of the VC dimension.

Corollary 4.3. *Let $\mathcal{H} \subset \{h\colon \mathcal{X} \to \{-1,1\}\}$. Then, for every $\delta > 0$, with probability at least $1-\delta$ for any $h \in \mathcal{H}$:*

$$\mathcal{R}(h) \leq \widehat{\mathcal{R}}_S(h) + \sqrt{\frac{2d\log(\frac{em}{d})}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}},$$

where d is the VC dimension of $\mathcal{H}$ and $m \geq d$.

5 Lecture 5 - The Mysterious Machine

Having established some theory, we are now ready for the first challenge.

[1]:
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sn
%matplotlib inline
```

Two files will be supplied to you via Moodle. A test and training set 'data_train_db.csv' and 'data_test_db.csv'. They were taken by observing a mystery machine. The first entry 'Running' is 1 if the machine worked. It is 0 if it failed to work. In the test set, the labels, are set to 2. You should predict them.

Let us look at out data first:

[2]:
```
data_train_db = pd.read_csv('data_train_db.csv')
data_test_db = pd.read_csv('data_test_db.csv')
data_train_db.head()
```

[2]:
```
   Running  Blue Switch On  Battery level  Humidity  Magnetic field
0      1.0             1.0       0.504463  0.654691        0.809938
1      1.0             1.0       0.441385  0.597252        0.690019
2      0.0             1.0       0.497714  0.521752        0.512899
3      0.0             0.0       0.729477  0.974705        0.629772
4      0.0             1.0       0.828015  0.768117        0.694428

[5 rows x 100 columns]
```

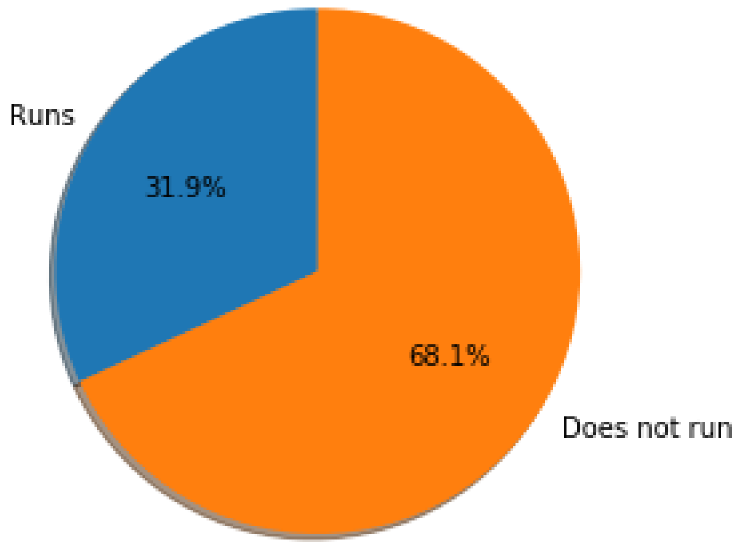
Lets look at some more properties of the data:

[3]:
```
data_train_db.describe()
```

[3]:
```
           Running  Blue Switch On  Battery level     Humidity
count  2000.000000     2000.000000    2000.000000  2000.000000
mean      0.319000        0.803436       0.697403     0.699631
std       0.466206        1.344869       1.604714     0.903394
min       0.000000      -42.078674     -54.697685   -29.500793
25%       0.000000        1.000000       0.556451     0.556232
50%       0.000000        1.000000       0.706002     0.699358
75%       1.000000        1.000000       0.853678     0.852918
max       1.000000       18.242558      44.936291    25.747851
```

How is the distribution of the labels?

```
[4]: data_train = data_train_db.values

labels = 'Runs', 'Does not run'
sizes = [np.sum(data_train[:,0]), np.sum(1-data_train[:,0])]

fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```

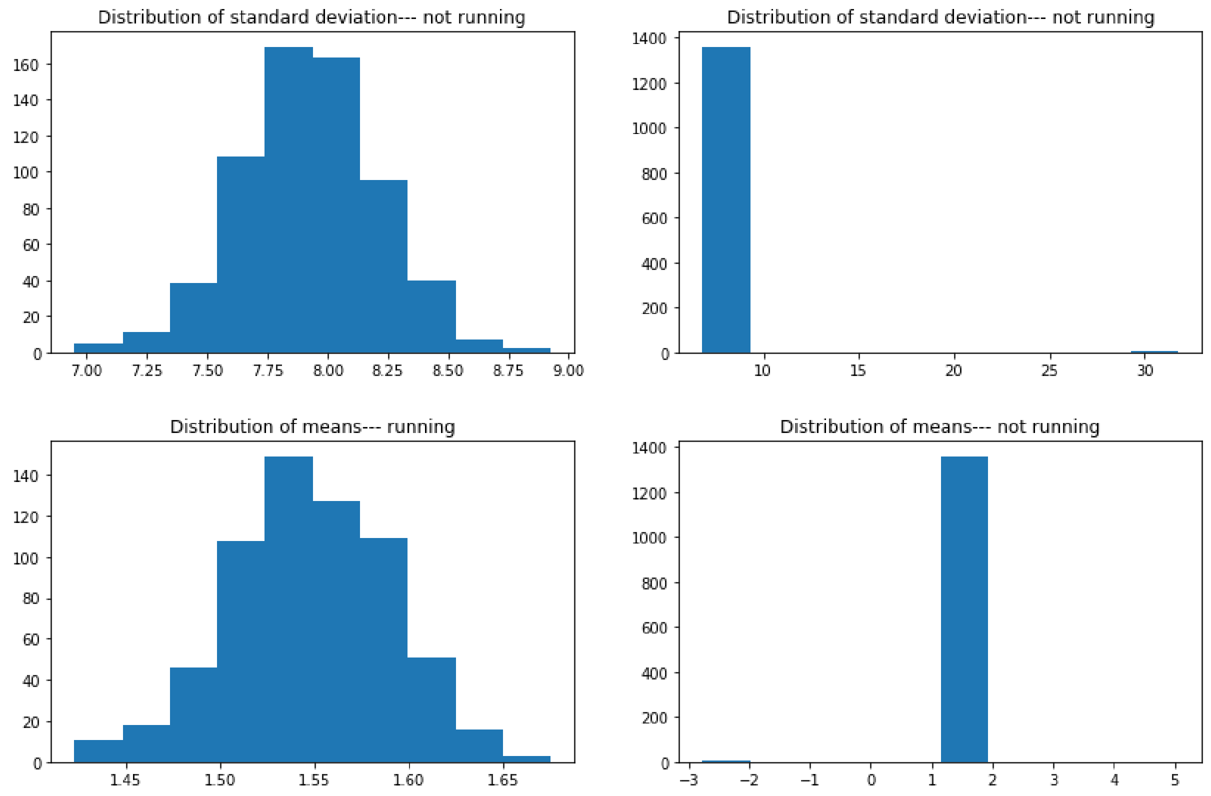


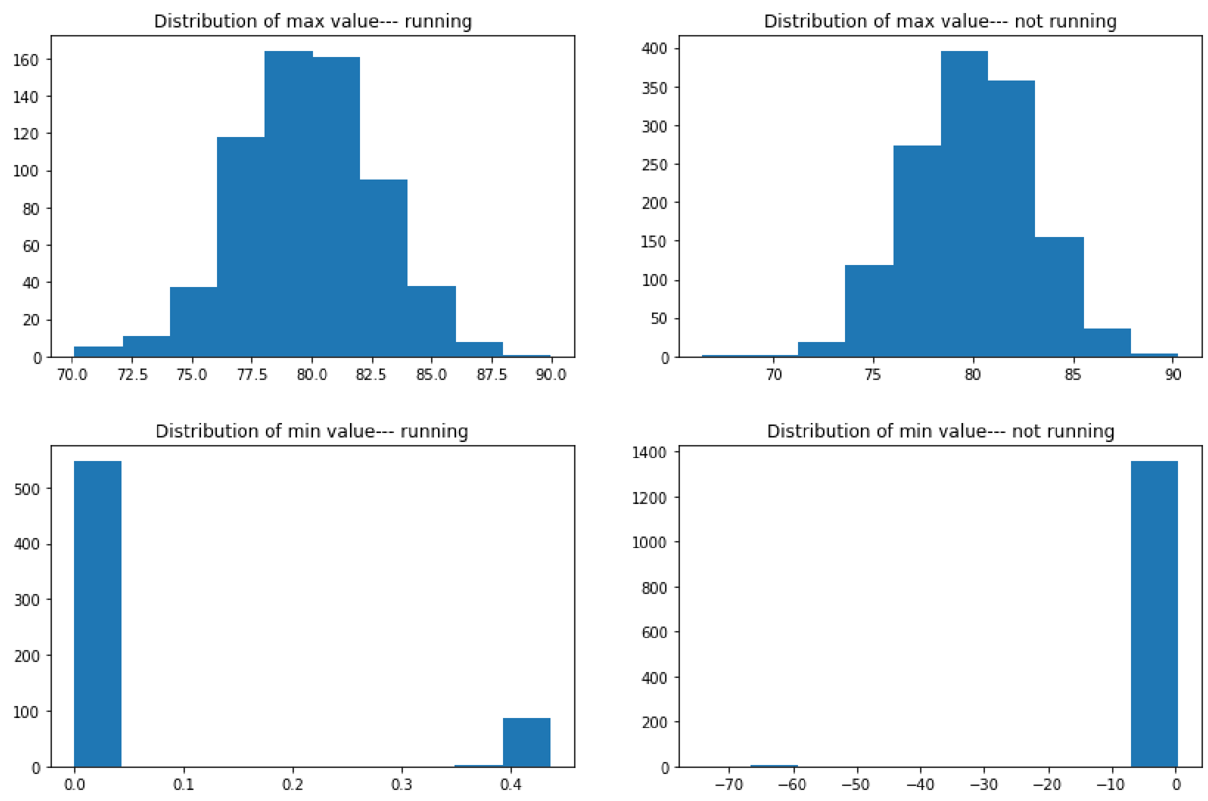
Lets look at some standard statistics of the data:

```
[5]: fig = plt.figure(figsize = (14, 4))

plt.subplot(1,2,1)
plt.hist(data_train[data_train[:,0] == 1,1:].std(1))
plt.title('Distribution of standard deviation--- not running')
plt.subplot(1,2,2)
plt.hist(data_train[data_train[:,0] == 0,1:].std(1))
plt.title('Distribution of standard deviation--- not running')

fig = plt.figure(figsize = (14, 4))

plt.subplot(1,2,1)
plt.hist(np.sum(data_train[data_train[:,0]==1,1:], 1)/100)
plt.title('Distribution of means--- running')
plt.subplot(1,2,2)
plt.hist(np.sum(data_train[data_train[:,0]==0,1:], 1)/100)
plt.title('Distribution of means--- not running')
```

```python
fig = plt.figure(figsize = (14, 4))

plt.subplot(1,2,1)
plt.hist(np.amax(data_train[data_train[:,0] == 1,1:], axis = 1))
plt.title('Distribution of max value--- running')
plt.subplot(1,2,2)
plt.hist(np.amax(data_train[data_train[:,0] == 0,1:], axis = 1))
plt.title('Distribution of max value--- not running')

fig = plt.figure(figsize = (14, 4))

plt.subplot(1,2,1)
plt.hist(np.amin(data_train[data_train[:,0] == 1,1:], axis = 1))
plt.title('Distribution of min value--- running')
plt.subplot(1,2,2)
plt.hist(np.amin(data_train[data_train[:,0] == 0,1:], axis = 1))
plt.title('Distribution of min value--- not running')
```

[5]:
```
Text(0.5, 1.0, 'Distribution of min value--- not running')
```





The distribution of the min values is a bit worrying. Since some very few entries have very high standard deviation, some very few and possibly the same values have very low negative values, but almost all other entries have only positive values. This may be a problem in the data set. We decide that these entries are outliers and drop these entries from the data base.

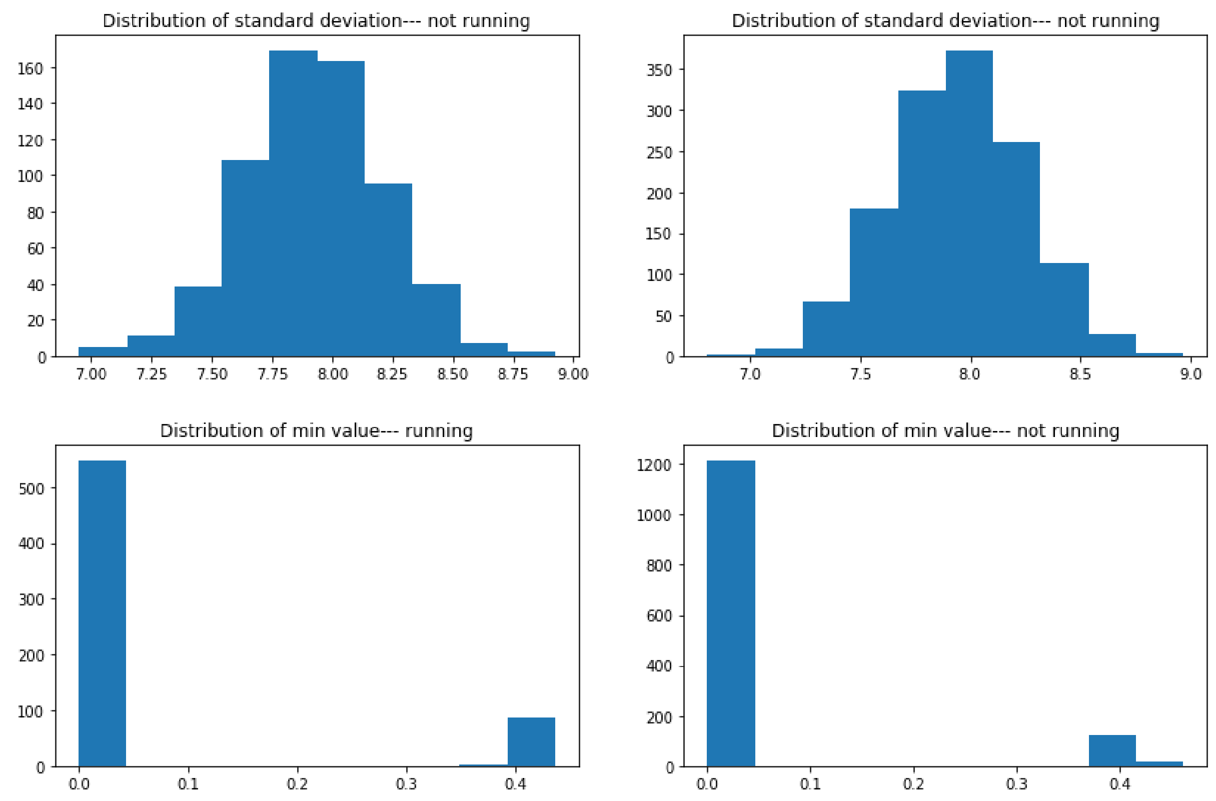
[6]:
```
# It seems like there are some data points which have much higher standard deviation than most. Let us␣
↪just remove those.


def clean_dataset(data):
    to_drop= []
    for k in range(data.shape[0]):
        if data[k,:].std()>15:
            to_drop.append(k)
    return np.delete(data, to_drop, axis = 0)
```

Let us apply the cleaning and look at the data set again

[7]:
```
data_train = clean_dataset(data_train)

fig = plt.figure(figsize = (14, 4))

plt.subplot(1,2,1)
plt.hist(data_train[data_train[:,0] == 1,1:].std(1))
plt.title('Distribution of standard deviation--- not running')
plt.subplot(1,2,2)
plt.hist(data_train[data_train[:,0] == 0,1:].std(1))
plt.title('Distribution of standard deviation--- not running')
```

```
fig = plt.figure(figsize = (14, 4))

plt.subplot(1,2,1)
plt.hist(np.amin(data_train[data_train[:,0] == 1,1:], axis = 1))
plt.title('Distribution of min value--- running')
plt.subplot(1,2,2)
plt.hist(np.amin(data_train[data_train[:,0] == 0,1:], axis = 1))
plt.title('Distribution of min value--- not running')
```

[7]:
```
Text(0.5, 1.0, 'Distribution of min value--- not running')
```



This looks much better.

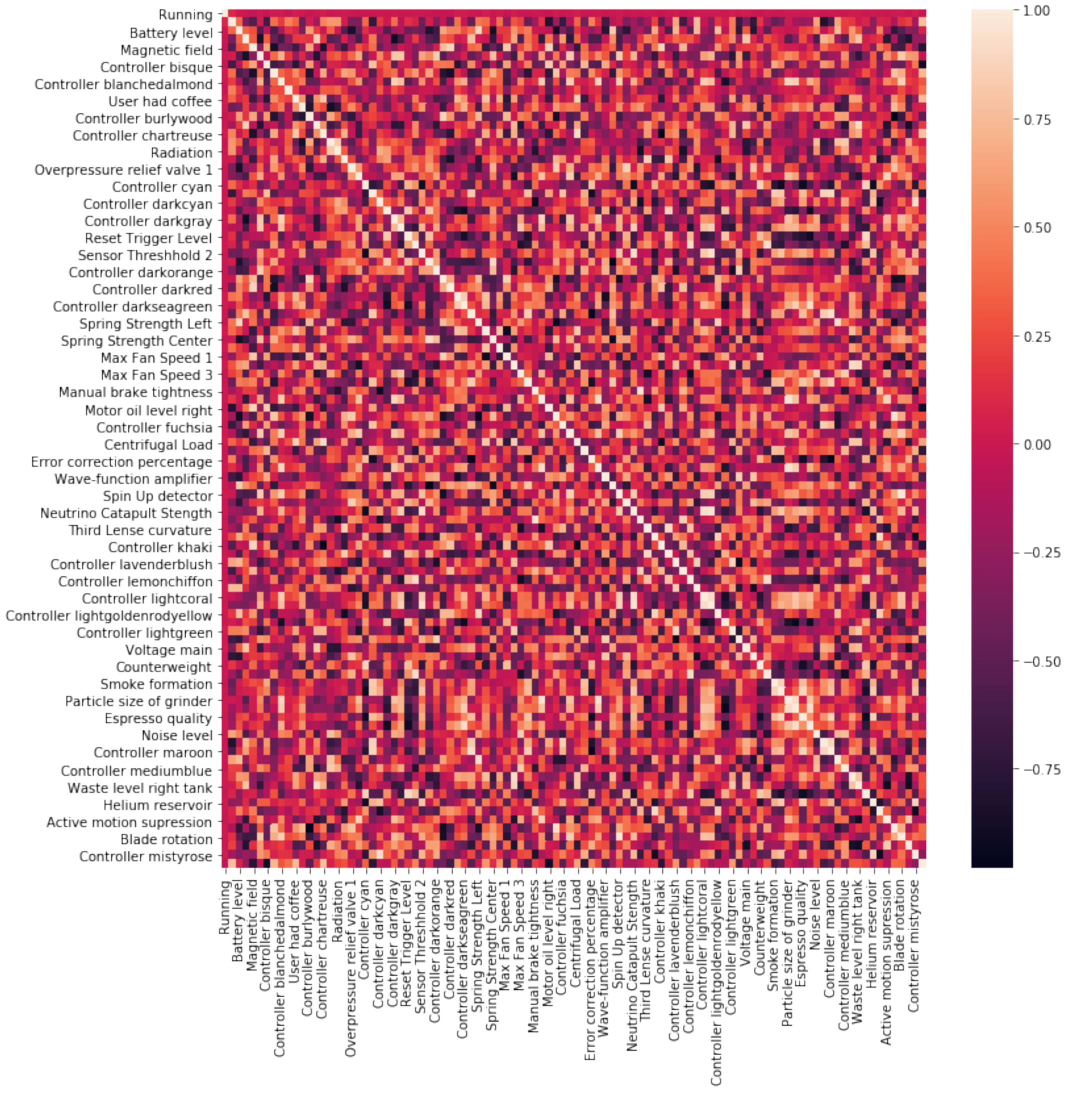
Now we start understanding our data set a bit more in detail. Let us try to get a feeling of the dependencies between the columns.

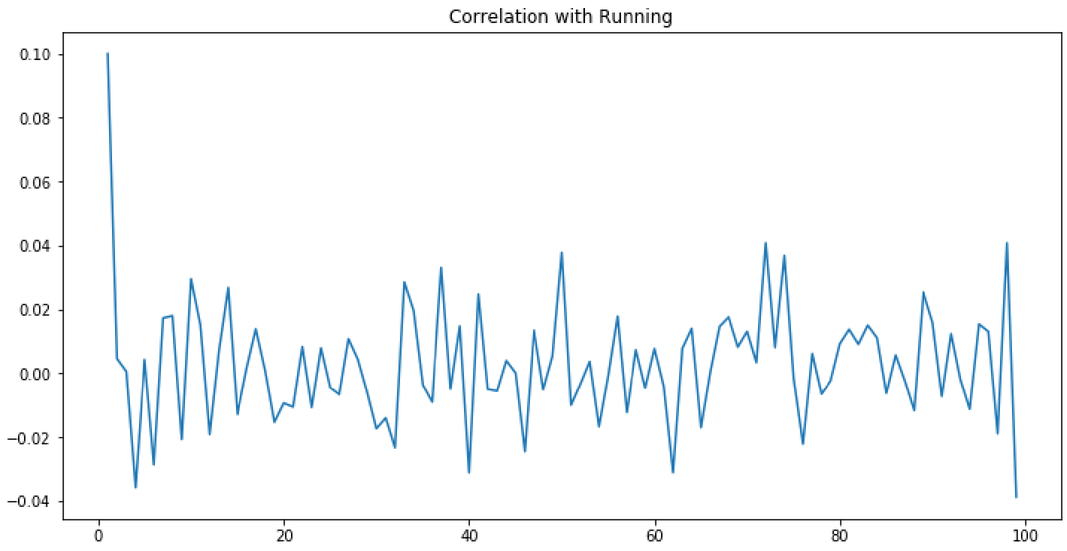
[8]:
```
data_train_db.corr()
```

[8]:
```
                 Running  Blue Switch On  Battery level  Humidity
Running         1.000000        0.100058       0.004500  0.000527
Blue Switch On  0.100058        1.000000       0.373730 -0.374582
Battery level   0.004500        0.373730       1.000000  0.353327
Humidity        0.000527       -0.374582       0.353327  1.000000
Magnetic field -0.035802       -0.554634       0.272756  0.321244
...                  ...             ...            ...       ...
Blade density   0.015307       -0.144248      -0.737770 -0.151252
Blade rotation  0.012993        0.148190      -0.527079 -0.675620
```

```
Controller mintcream -0.018974        0.208065      -0.278604  -0.801354
Controller mistyrose  0.040784       -0.166662      -0.335028  -0.225535
Controller moccasin  -0.038704        0.771165       0.072201  -0.240072

[100 rows x 100 columns]
```

[9]:
```python
corrMatrix = data_train_db.corr()
plt.figure(figsize = (12,12))
sn.heatmap(corrMatrix, annot=False)
plt.show()

plt.figure(figsize = (12,6))
plt.plot(np.arange(1, 100), corrMatrix['Running'][1:100])
plt.title('Correlation with Running')
plt.show()
```

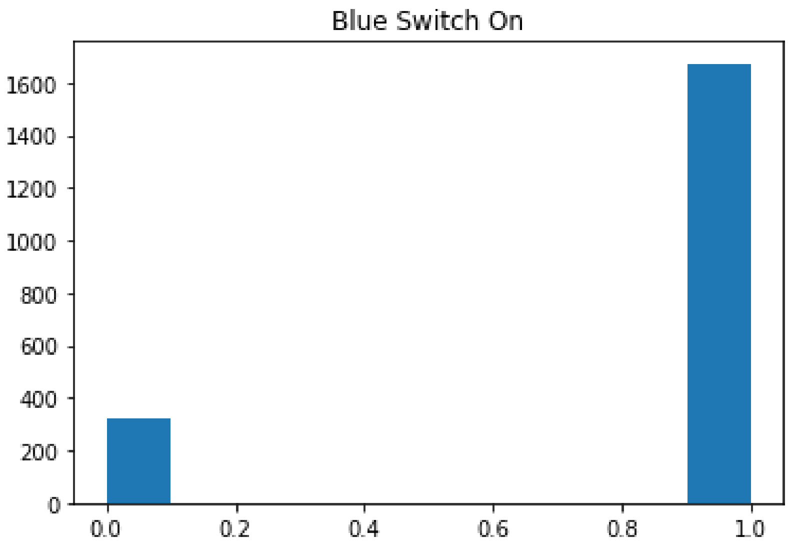




The first row of the data set (after the row 'Running' itself), seems to be suspiciously important. Lets look at it in isolation.

```
[10]: plt.hist(data_train[:,1])
      plt.title(data_train_db.columns[1])
```

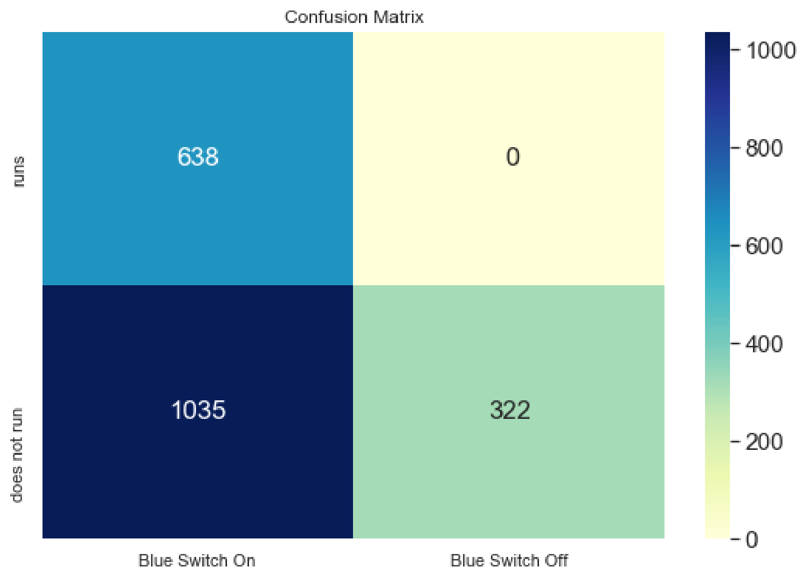
```
[10]: Text(0.5, 1.0, 'Blue Switch On')
```



We see that 'Blue Switch On' only takes two values (On and Off). Let us look in detail, what the effect of this switch is on whether the mechanism runs or not.

```
[16]: runs_switchon = np.count_nonzero((data_train[:,0]==1)*(data_train[:,1]==1))
      runs_switchoff = np.count_nonzero((data_train[:,0]==1)*(data_train[:,1]==0))
      runsnot_switchon = np.count_nonzero((data_train[:,0]==0)*(data_train[:,1]==1))
      runsnot_switchoff = np.count_nonzero((data_train[:,0]==0)*(data_train[:,1]==0))
      conf_matrix = [[runs_switchon, runs_switchoff], [runsnot_switchon, runsnot_switchoff]]

      sn.set(color_codes=True)
      plt.figure(1, figsize=(9, 6))
      plt.title("Confusion Matrix")

      sn.set(font_scale=1.4)
      ax = sn.heatmap(conf_matrix, annot=True, cmap="YlGnBu", fmt='2')

      ax.set_yticklabels(['runs', 'does not run'])
      ax.set_xticklabels(['Blue Switch On', 'Blue Switch Off'])
```

[16]:
```
[Text(0.5, 0, 'Blue Switch On'), Text(1.5, 0, 'Blue Switch Off')]
```



Now this is fantastic. If the Blue Switch is off, then the mechanism never works.

Next, we would like to extract additional important parameters of the machine. We rank the columns according to their correlation with ‘Running’:

[12]:
```
S = np.argsort(np.array(corrMatrix['Running']))[::-1]
print(S)
```

```
[ 0  1 72 98 50 74 37 10 33 14 89 41 34  8 56 68  7 90 95 11 83 39 67 64
 17 81 47 70 96 92 84 27 80 82 22 69 73 24 63 60 58 13 77 86 49  2 28  5
 44 53 71 16 18  3 66 45 55 75 93 79 87 52 35 61 25 59 38 42 48 43 29 85
 78 26 91 36 20 51 21 23 94 88 57 15 31 19 54 65 30 97 12  9 76 32 46  6
 62 40  4 99]
```

We saw that the first entry is always 1 if the method is working. Also from the ranking above, we expect that large values in coordinates 72 and 98 seem to indicate that the algorithm works.

Let us describe a hypothesis set that thakes this observation into account, by defining a classifier below. The hypothesis set is characterised by a threshholding value ‘thresh’.

[39]:
```
def myclassifier(data, thresh):
    if data[1] == 0:
        return 0 # If the blue switch is off, then we know that the mechanism wont work.
    if data[72] + data[98] > thresh:
        return 1
    return 0
```

Next we find the value thresh, that yields the best classification on the test set:

[40]:
```
best_thresh = 0
best_err = data_train.shape[0]

for tr in range(100):
    thresh = tr/20
    err = 0
    for t in range(data_train.shape[0]):
        err = err + (myclassifier(data_train[t, :], thresh) != data_train[t, 0])
    if err < best_err:
        best_err = err
        best_thresh = thresh

print('Test accuracy:' + str(1-best_err/data_train.shape[0]))
```

```
Test accuracy:0.7604010025062656
```

The test accuracy above is quite terrible. On the other hand, the hypothesis class seems very small, so Corollary 4.2 gives us some confidence that the result may generalise in the sense that it will not be worse on the test set. ("Not worse, but still very bad" is of course not a very desirable outcome.)

I am sure you can do much better than this.

[41]:
```
# Finally, we predict the result.
predicted_labels = np.zeros(data_test_db.shape[0])
data_test = data_test_db.values

for k in range(data_test_db.shape[0]):
    predicted_labels[k] = myclassifier(data_test[k, :], best_thresh)

np.savetxt('PhilippPetersens_prediction.csv', predicted_labels, delimiter=',')
```

Please send your result via email to philipp.petersen@univie.ac.at. Your email should include the names of all people who worked on your code, their student identification numbers, a name for your team, and the code used. It should also contain one or two paragraphs of a short description of the method you used.

6 Lecture 6 - Lower Bounds on Learning

A finite VC dimension guarantees a controllable generalisation error, but is it necessary? Yes!

Theorem 6.1. *Let $\mathcal{H}$ be a hypothesis set with* $\mathrm{VCdim}(\mathcal{H}) = d > 1$*. Then, for every $m \geq (d-1)/2$ and for every learning algorithm $\mathcal{A}$ there exists a distribution $\mathcal{D}$ over $\mathcal{X}$ and a target concept $g \in \mathcal{H}$ such that*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left[\mathcal{R}_{\mathcal{D}}(\mathcal{A}(S)) > \frac{d-1}{32m}\right] \geq 0.01.$$

Proof.

1. **Set-up:** We first build a very imbalanced distribution. Let $\overline{X} := \{x_1, x_2, \ldots, x_d\} \subset \mathcal{X}$ be a set that is shattered by $\mathcal{H}$.

   For $\epsilon > 0$, we define the distribution $\mathcal{D}_\epsilon$ by $\mathbb{P}(x_1) = 1 - 8\epsilon$ and $\mathbb{P}(x_k) = 8\epsilon/(d-1)$ for $k = 2, \ldots, d$.

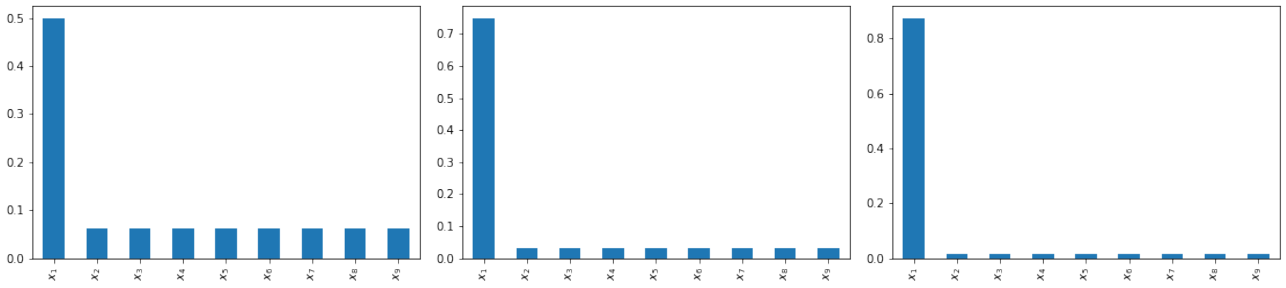


Figure 8: Distribution $\mathcal{D}_\epsilon$ for $\epsilon = 1/16, 1/32, 1/64$.

   For $S \in \mathcal{X}^m$, we denote $\overline{S} := \{s_i \in S \colon s_i \neq x_1 \text{ for all } i \in [m]\}$. Additionally, let $\mathcal{S} \subset \mathcal{X}^m$ be the set of samples, such that $|\overline{S}| \leq (d-1)/2$.

   Let, for $S \in \mathcal{X}^m$ and $u \in \{0,1\}^{d-1}$, $f_u \in \mathcal{H}$ be such that

$$f_u(x_1) = 1 \text{ and } f_u(x_k) = u_{k-1}.$$

   We have that f_u is well-defined since $\mathcal{H}$ shatters $\overline{X}$.

   Assume that $\mathcal{A}$ is any learning algorithm. We can assume without loss of generality that $\mathcal{A}(S)(x_1) = 1$. Otherwise, we could modify $\mathcal{A}$ to satisfy this and end up with a lower expected error since we will only consider concepts g below that satisfy $g(x_1) = 1$.

2. **Bounding the expected error for a fixed sample:**

   Let $\mathcal{U}$ be a uniform distribution on $\{0,1\}^{d-1}$, then for any $S \in \mathcal{X}^m$,

$$\mathbb{E}_{\mathcal{U}}(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{\mathcal{U}})) = \sum_{u \in \{0,1\}^{d-1}} \sum_{k=2}^{d} \mathbb{1}_{\mathcal{A}(S)(x_k) \neq f_u(x_k)} \mathbb{P}[x_k] \mathbb{P}[u],$$

   where $\mathbb{E}(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_u))$ denotes the expected risk with target concept f_u. By reducing the set that we sum over, we may estimate from below by

$$\begin{aligned} \mathbb{E}_{\mathcal{U}}(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{\mathcal{U}})) &\geq \sum_{u \in \{0,1\}^{d-1}} \sum_{\substack{k=2 \\ x_k \notin \overline{S}}}^{d} \mathbb{1}_{\mathcal{A}(S)(x_k) \neq f_u(x_k)} \mathbb{P}[x_k] \mathbb{P}[u] \\ &= \sum_{\substack{k=2 \\ x_k \notin \overline{S}}}^{d} \left(\sum_{u \in \{0,1\}^{d-1}} \mathbb{1}_{\mathcal{A}(S)(x_k) \neq f_u(x_k)} \mathbb{P}[u] \right) \mathbb{P}[x_k]. \end{aligned}$$

   Per definition of f_u it is clear that for every x_k, where $k > 1$ it holds that $\mathbb{1}_{\mathcal{A}(S)(x_k) = f_u(x_k)} = 1$ on exactly half of all values $u \in \{0,1\}^{d-1}$. Hence, we estimate that

$$\mathbb{E}_{\mathcal{U}}(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{\mathcal{U}})) \geq \sum_{\substack{k=2 \\ k \notin \overline{S}}}^{d} \frac{1}{2} \mathbb{P}[x_k] = \frac{1}{2} \left(d - 1 - |\overline{S}| \right) \frac{8\epsilon}{d-1}.$$

Thus, if $S \in \mathcal{S}$, then

$$\mathbb{E}_{\mathcal{U}}(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{\mathcal{U}})) \geq \sum_{\substack{k=2 \\ x_k \notin \overline{S}}}^{d} \frac{1}{2}\mathbb{P}[x_k] \geq \frac{1}{2}\left(\frac{d-1}{2}\right)\frac{8\epsilon}{d-1} = 2\epsilon. \tag{15}$$

3. **Finding one 'bad' concept:**

   We conclude from (15) that

   $$\mathbb{E}_{S\in\mathcal{S}}\mathbb{E}_{\mathcal{U}}(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{\mathcal{U}})) \geq 2\epsilon.$$

   By Fubini's theorem, we also have that

   $$\mathbb{E}_{\mathcal{U}}(\mathbb{E}_{S\in\mathcal{S}}\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{\mathcal{U}}) \geq 2\epsilon. \tag{16}$$

   The estimate on the expected value (16) implies that there exists at least one $u^* \in \{0,1\}^{d-1}$ such that

   $$\mathbb{E}_{S\in\mathcal{S}}\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*}) \geq 2\epsilon. \tag{17}$$

   Note that, for every $S \in \mathcal{X}^m$

   $$\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*}) = \sum_{k=2}^{d} \mathbb{1}_{\mathcal{A}(S)(x_k)\neq f_{u^*}(x_k)}\mathbb{P}[x_k] \leq \sum_{k=2}^{d}\frac{8\epsilon}{d-1} = 8\epsilon. \tag{18}$$

   Now we can compute for

   $$\begin{aligned}
   \mathbb{E}_{S\in\mathcal{S}}\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*}) &= \sum_{S:\, \mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*})\geq\epsilon} \mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*})\mathbb{P}(S|\mathcal{S}) \\
   &\quad + \sum_{S:\, \mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*})<\epsilon} \mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*})\mathbb{P}(S|\mathcal{S}) \\
   &\overset{(18)}{\leq} \sum_{S:\, \mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*})\geq\epsilon} 8\epsilon\, \mathbb{P}(S|\mathcal{S}) \\
   &\quad + \sum_{S:\, \mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*})<\epsilon} \epsilon\, \mathbb{P}(S|\mathcal{S}) \\
   &\leq 8\epsilon\, \mathbb{P}(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*}) \geq \epsilon) + \epsilon\,(1 - \mathbb{P}(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*}) \geq \epsilon)) \\
   &= \epsilon + 7\epsilon\, \mathbb{P}(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*}) \geq \epsilon)).
   \end{aligned}$$

   With (17), we conclude that if $S \in \mathcal{S}$

   $$\mathbb{P}(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*}) \geq \epsilon)) \geq \frac{1}{7}.$$

   More generally, for arbitrary $S \sim \mathcal{D}_\epsilon$ we have that

   $$\mathbb{P}_{S\sim\mathcal{D}_\epsilon}(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*}) \geq \epsilon)) \geq \frac{\mathbb{P}_{\mathcal{D}_\epsilon}(\mathcal{S})}{7}. \tag{19}$$

4. **Find $\mathbb{P}_{\mathcal{D}_\epsilon}[\mathcal{S}]$:**

   We will use the following *multiplicative Chernoff bound:*

Theorem 6.2 (Multiplicative Chernoff Bound)**.** *Let $X_1, \dots, X_m$ be independent random variables drawn according to a distribution $\mathcal{D}$ with mean μ and such that $0 \le X_k \le 1$ almost surely for all $k \in [m]$. Then, for $\gamma \in [0, 1/\mu - 1]$ it holds that*

$$\mathbb{P}[\overline{\mu} \ge (1+\gamma)\mu] \le e^{-\frac{m\mu\gamma^2}{3}}$$
$$\mathbb{P}[\overline{\mu} \le (1-\gamma)\mu] \le e^{-\frac{m\mu\gamma^2}{2}},$$

where $\overline{\mu} = \frac{1}{m}\sum_{i=1}^m X_i$.

Let $Y_1, \dots, Y_m$ be i.i.d distributed as $\mathcal{D}_\epsilon$. Further let for $k \in [m]$

$$Z_k := \mathbb{1}_{\{x_2,\dots,x_d\}}(Y_k).$$

It is clear that $\mathbb{E}(Z_k) = 8\epsilon$. Assuming that $8\epsilon \le 1/2$, we can apply Theorem 6.2 with $\gamma = 1$ to obtain

$$\mathbb{P}\left(\sum_{i=1}^m Z_i \ge 16\epsilon m\right) \le e^{-\frac{8\epsilon m}{3}}. \tag{20}$$

Now notice that if a sample $S = (Y_1, \dots, Y_m)$ is not in $\mathcal{S}$ then the associated $(Z_1, \dots, Z_m)$ must satisfy $\sum_{i=1}^m Z_i > (d-1)/2$. Therefore,

$$1 - \mathbb{P}(\mathcal{S}) \le \mathbb{P}\left(\sum_{i=1}^m Z_i \ge (d-1)/2\right).$$

5. **Finishing the proof:** Setting $\epsilon = (d-1)/(32m) \le 1/16$, we conclude that

$$\mathbb{P}(\mathcal{S}) \ge 1 - e^{-\frac{d-1}{12}} \ge 7\delta,$$

for $\delta > 1/100$.

We conclude with (19), that

$$\mathbb{P}\left(\mathcal{R}_{\mathcal{D}_\epsilon}(\mathcal{A}(S), f_{u^*}) \ge \frac{d-1}{32m}\right) > \frac{1}{100},$$

which is the claim.

$\square$

A similar result to Theorem 6.1 holds in the non-realisable/agnostic setting.

Theorem 6.3. *Let $\mathcal{H}$ be a hypothesis set with $d = \mathrm{VCdim}(\mathcal{H}) > 1$. Then for $m \in \mathbb{N}$ and any learning algorithm $\mathcal{A}$, there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \{-1, 1\}$ such that*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left(\mathcal{R}_{\mathcal{D}}(\mathcal{A}(S)) - \inf_{h\in\mathcal{H}} \mathcal{R}_{\mathcal{D}}(h) > \sqrt{\frac{d}{320m}}\right) \ge \frac{1}{64}.$$

7 Lecture 7 - The Mysterious Machine - Discussion

This will be a discussion about the challenge as well as help with coding issues.

I recommend that you should use Python and the Jupyter notebook. See, for example `https://jupyter.org/install` for a guide to install both.

8 Lecture 8 - Model Selection

Ho do we choose an appropriate hypothesis set or learning algorithm for a given problem?

For a given binary hypothesis class $\mathcal{H}$ and a function $h \in \mathcal{H}$, we have that

$$\mathcal{R}(h) - R^* = \underbrace{\left(\mathcal{R}(h) - \inf_{g\in\mathcal{H}} \mathcal{R}(g)\right)}_{\text{estimation}} + \underbrace{\left(\inf_{g\in\mathcal{H}} \mathcal{R}(g) - R^*\right)}_{\text{approximation}}. \tag{21}$$

where R^* is the Bayes error of Definition 3.2. See Figure 9 for a visualisation of (21).

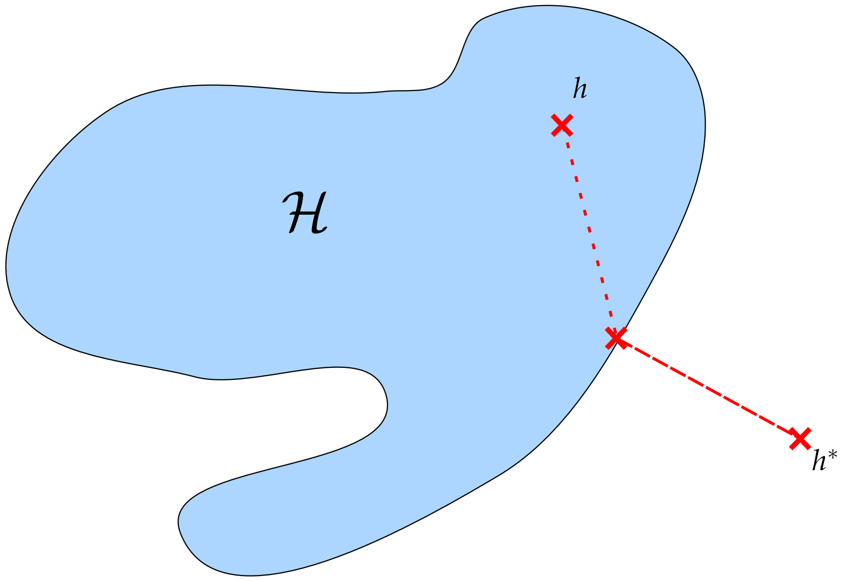


Figure 9: Visualisation of (21), where h^* is the Bayes classifier.

8.1 Empirical Risk Minimisation

Empirical risk minimisation is the algorithm that chooses the hypothesis with the smallest empirical risk.

Definition 8.1. *Let $\mathcal{H}$ be a hypothesis set, S be a sample then we define the solution of* empirical risk minimisation

$$h_S^{ERM} := \arg\min_{h\in\mathcal{H}} \widehat{\mathcal{R}}_S(h).$$

Note, that h_S^{ERM} does not need to exist, but if S is finite and $\mathcal{Y}$ is too, as in the binary classification case, then it is easy to see that h_S^{ERM} is well defined.

We have that the empirical risk minimiser inflicts a small estimation error if the generalisation error is small.

Proposition 8.1. *Let $\mathcal{H}$ be a hypothesis set, S be a sample. Then we have that*

$$\mathbb{P}\left(\mathcal{R}(h_S^{ERM}) - \inf_{h\in\mathcal{H}} \mathcal{R}(h) > \epsilon\right) \leq \mathbb{P}\left(\sup_{h\in\mathcal{H}} |\mathcal{R}(h) - \widehat{\mathcal{R}}_S(h)| > \epsilon/2\right). \tag{22}$$

Proof. For every $\delta > 0$, there exists $h_\delta \in \mathcal{H}$ such that $\mathcal{R}(h_\delta) - \inf_{h\in\mathcal{H}} \mathcal{R}(h) < \delta$. Therefore, we have that

$$\mathbb{P}\left(\mathcal{R}(h_S^{ERM}) - \inf_{h\in\mathcal{H}} \mathcal{R}(h) > \epsilon\right) \leq \mathbb{P}\left(\mathcal{R}(h_S^{ERM}) - \mathcal{R}(h_\delta) > \epsilon - \delta\right), \tag{23}$$

for all $\delta > 0$.

Moreover,

$$\begin{aligned}\mathcal{R}(h_S^{ERM}) - \mathcal{R}(h_\delta) &= \mathcal{R}(h_S^{ERM}) - \widehat{\mathcal{R}}_S(h_S^{ERM}) + \widehat{\mathcal{R}}_S(h_S^{ERM}) - \mathcal{R}(h_\delta)\\ &\leq \mathcal{R}(h_S^{ERM}) - \widehat{\mathcal{R}}_S(h_S^{ERM}) + \widehat{\mathcal{R}}_S(h_\delta) - \mathcal{R}(h_\delta)\\ &\leq 2\sup_{h\in\mathcal{H}} |\mathcal{R}(h) - \mathcal{R}(h_h)|.\end{aligned}$$

We obtain from (23) that

$$\mathbb{P}\left(\mathcal{R}(h_S^{ERM}) - \inf_{h\in\mathcal{H}} \mathcal{R}(h) > \epsilon\right) \leq \mathbb{P}\left(\sup_{h\in\mathcal{H}} |\mathcal{R}(h) - \mathcal{R}(h_h)| > \frac{\epsilon-\delta}{2}\right). \tag{24}$$

Since the left hand side of (24) is independent from δ we obtain the claim from the continuity of measures.

□

We saw before that we can control the right hand side of (22), if the VC dimension of $\mathcal{H}$ is bounded. Thereby, (22) yields a bound on the estimation error. However, requiring a small VC dimension does not let us take a very large hypothesis space. This implies that we may have a large approximation error.

8.2 Structural risk minimisation

Here we perform ERM over nested hypothesis spaces

$$\mathcal{H}_1 \subset \mathcal{H}_2 \subset \cdots \subset \mathcal{H}_k \subset \cdot\ldots$$

The approximation error will decrease (or at least not increase) for growing k, while the estimation decreases with decreasing k. The idea is shown in Figure 10.

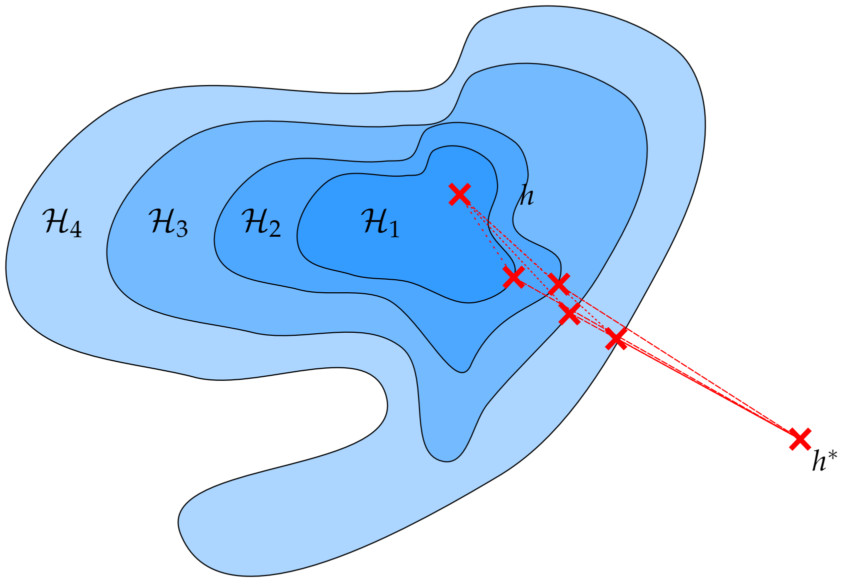


Figure 10: Visualisation of structural risk minimisation, where h^* is the Bayes classifier.

Structural risk minimisation is a method to choose an appropriate value of k. Here one employs a penalty on large terms.

Definition 8.2. *Let $(\mathcal{H}_k)_{k=1}^{\infty}$ be a sequence of hypothesis sets and let S be a sample. Then, the solution of* structural risk minimisation *is*

$$h_S^{SRM} := \arg\min\{F_k(h)\colon k \in \mathbb{N}, h \in \mathcal{H}_k\},$$

where

$$F_k(h) := \widehat{\mathcal{R}}_S(h) + \mathfrak{R}_m(\mathcal{H}_k) + \sqrt{\frac{\log k}{m}}.$$

We have the following learning guarantee for SRM:

Theorem 8.1. *Let $\delta > 0$, $(\mathcal{H}_k)_{k=1}^{\infty}$ be a sequence of hypothesis sets, $\mathcal{H} := \bigcup_{k\in\mathbb{N}} \mathcal{H}_k$, and let $\mathcal{D}$ be a distribution. With probability at least $1-\delta$ for a sample $S \sim \mathcal{D}^m$, it holds that*

$$\mathcal{R}(h_S^{SRM}) \leq \inf_{h\in\mathcal{H}} \left(\mathcal{R}(h) + 2\mathfrak{R}_m(\mathcal{H}_{k(h)}) + \sqrt{\frac{\log k(h)}{m}}\right) + \sqrt{\frac{2\log(3/\delta)}{m}},$$

where $k(h)$ is the smallest k such that $h \in \mathcal{H}_k$.

Proof. We first remind ourselves of Theorem 4.1, where we found that with probability at least $1-\delta$

$$\mathcal{R}(h) \leq \widehat{\mathcal{R}}_S(h) + \mathfrak{R}_m(\mathcal{H}) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}$$

or equivalently

$$\mathbb{P}\left(\mathcal{R}(h) - \widehat{\mathcal{R}}_S(h) - \mathfrak{R}_m(\mathcal{H}) > \delta\right) \leq e^{-2m\delta^2}. \tag{25}$$

We compute with a union bound that

$$\begin{aligned}\mathbb{P}\left(\sup_{h\in\mathcal{H}} \mathcal{R}(h) - F_{k(h)}(h) > \epsilon\right) &= \mathbb{P}\left(\sup_{k\in\mathbb{N}} \sup_{h\in\mathcal{H}_k} \mathcal{R}(h) - F_k(h) > \epsilon\right) \\ &\leq \sum_{k=1}^{\infty} \mathbb{P}\left(\sup_{h\in\mathcal{H}_k} \mathcal{R}(h) - F_k(h) > \epsilon\right).\end{aligned}$$

Invoking the definition of F_k and (25) yields that

$$\begin{aligned}&\sum_{k=1}^{\infty} \mathbb{P}\left(\sup_{h\in\mathcal{H}_k} \mathcal{R}(h) - F_k(h) > \epsilon\right) \\ &\quad= \sum_{k=1}^{\infty} \mathbb{P}\left(\sup_{h\in\mathcal{H}_k} \mathcal{R}(h) - \widehat{\mathcal{R}}_S(h) - \mathfrak{R}_m(h) > \epsilon + \sqrt{\log(k)/m}\right) \\ &\quad\leq \sum_{k=1}^{\infty} \exp\left(-2m\left(\epsilon + \sqrt{\log(k)/m}\right)^2\right) \\ &\quad\leq \sum_{k=1}^{\infty} e^{-2m\epsilon^2} e^{-2\log(k)} \\ &\quad= e^{-2m\epsilon^2} \sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} e^{-2m\epsilon^2} \leq 2e^{-2m\epsilon^2}.\end{aligned} \tag{26}$$

Consider two random variables X_1, X_2. It is clear that for every $t \in \mathbb{R}$

$$\mathbb{P}(X_1 + X_2 > t) \leq \mathbb{P}(X_1 > t/2) + \mathbb{P}(X_2 > t/2). \tag{27}$$

Now we compute for an arbitrary $h \in \mathcal{H}$

$$\begin{aligned}
&\mathbb{P}\left(\mathcal{R}(h_S^{SRM}) - \mathcal{R}(h) - 2\mathfrak{R}_m(\mathcal{H}_{k(h)}) - \sqrt{\frac{\log k(h)}{m}} > \epsilon\right) \\
\text{[Equation (27)]} \quad &\leq \mathbb{P}\left(\mathcal{R}(h_S^{SRM}) - F_{k(h_S^{SRM})}(h_S^{SRM}) > \epsilon/2\right) \\
&\quad + \mathbb{P}\left(F_{k(h_S^{SRM})}(h_S^{SRM}) - \mathcal{R}(h) - 2\mathfrak{R}_m(\mathcal{H}_{k(h)}) - \sqrt{\frac{\log k(h)}{m}} > \epsilon/2\right) \\
\text{[Equation (26)]} \quad &\leq 2e^{-m\epsilon^2/2} + \mathbb{P}\left(F_{k(h_S^{SRM})}(h_S^{SRM}) - \mathcal{R}(h) - 2\mathfrak{R}_m(\mathcal{H}_{k(h)}) - \sqrt{\frac{\log k(h)}{m}} > \epsilon/2\right) \\
&\leq 2e^{-m\epsilon^2/2} + \mathbb{P}\left(F_{k(h)}(h) - \mathcal{R}(h) - 2\mathfrak{R}_m(\mathcal{H}_{k(h)}) - \sqrt{\frac{\log k(h)}{m}} > \epsilon/2\right) \\
&\leq 2e^{-m\epsilon^2/2} + \mathbb{P}\left(\widehat{\mathcal{R}}_S(h) - \mathcal{R}(h) - \mathfrak{R}_m(\mathcal{H}_{k(h)}) > \epsilon/2\right) \\
\text{[Equation (25)]} \quad &\leq 3e^{-m\epsilon^2/2}.
\end{aligned}$$

This completes the proof. □

Remark 8.1. *Except for the term* $\sqrt{\log(k(h))/m}$ *the generalisation bound of SRM is that of the best hypothesis from the sequence* $(\mathcal{H})_{k=1}^{\infty}$. *On the flip side, we would need to solve many empirical risk minimisations and know the Rademacher complexities of all individual hypothesis sets.*

8.3 Cross-validation

Definition 8.3. *Let* $(\mathcal{H}_k)_{k=1}^{\infty}$ *be sequence of hypothesis sets. Let* $\alpha \in (0,1)$ *and* $S = (x_i, y_i)_{i=1}^m$ *be a sample. Then, the solution of* cross-validation *is*

$$h_S^{CV} := \arg\min\{\widehat{\mathcal{R}}_{S_2}(h_{S_1,k}^{ERM}) \colon h_{S_1,k}^{ERM} \in \mathcal{H}_k, k \in \mathbb{N}\},$$

where $S^1 = (x_i, y_i)_{i=1}^{m'}$ *for* $m' = \lceil (1-\alpha)m \rceil$, $S_2 = (x_i, y_i)_{i=m'+1}^m$, *and* $h_{S_1,k}^{ERM}$ *is the empirical risk minimiser over the hypothesis class* $\mathcal{H}_k$ *with sample* S_1.

In words, cross-validation consists in setting aside a validation set on which the loss is measured, but which is not used for training.

The following proposition will be stated without proof. It shows that with high probability, the empirical risk with respect to S_2 is close to the expected risk. The proof is based on Hoeffding's inequality.

Proposition 8.2. *Let* $(\mathcal{H}_k)_{k=1}^{\infty}$ *be sequence of hypothesis sets. Let* $\alpha \in (0,1)$, *and let* $S \sim \mathcal{D}^m$ *and* S_1 *be as in Definition 8.3, then it holds that*

$$\mathbb{P}\left(\sup_{k \geq 1} \left|\mathcal{R}(h_{S_1,k}^{ERM}) - \widehat{\mathcal{R}}_{S_2}(h_{S_1,k}^{ERM})\right| - \sqrt{\frac{\log k}{\alpha m}} > \epsilon\right) \leq 4e^{-2\alpha m\epsilon^2}. \tag{28}$$

Based on the result above, we can show that cross-validation can often perform very similarly to structural risk minimisation.

Theorem 8.2. *Let $(\mathcal{H}_k)_{k=1}^{\infty}$ be sequence of hypothesis sets. Let $\alpha \in (0,1)$, and let $S \sim \mathcal{D}^m$ and S_1 be as in Definition 8.3. For every $\delta \in (0,1)$ it holds with probability $1-2\delta$ that*

$$\mathcal{R}(h_S^{CV}) - \mathcal{R}(h_{S_1}^{SRM}) \leq 2\sqrt{\frac{\log(\max\{k(h_S^{CV}), k(h_{S_1}^{SRM})\})}{\alpha m}} + 2\sqrt{\frac{\log(4/\delta)}{2\alpha m}},$$

where $k(h)$ denotes the smallest k such that $h \in \mathcal{H}_k$.

Proof. We have by Proposition 8.2 that

$$\mathcal{R}(h_S^{CV}) \leq \widehat{\mathcal{R}}_{S_2}(h_S^{CV}) + \sqrt{\frac{\log(k(h_S^{CV}))}{\alpha m}} + \sqrt{\frac{\log(4/\delta)}{2\alpha m}} =: \mathrm{I}, \tag{29}$$

with probability $1-\delta$. Since $h_{S_1}^{SRM}$ is an empirical risk minimiser on $\mathcal{H}_k$ with $k = k(h_{S_1}^{SRM})$, we have that $\widehat{\mathcal{R}}_{S_2}(h_S^{CV}) \leq \widehat{\mathcal{R}}_{S_2}(h_{S_1}^{SRM})$ and hence

$$\mathrm{I} \leq \widehat{\mathcal{R}}_{S_2}(h_{S_1}^{SRM}) + \sqrt{\frac{\log(k(h_S^{CV}))}{\alpha m}} + \sqrt{\frac{\log(4/\delta)}{2\alpha m}} =: \mathrm{II}$$

Again, for $k = k(h_{S_1}^{SRM})$ it holds that $h_{S_1}^{SRM}$ is the empirical risk minimiser over $\mathcal{H}_k$. Therefore, we get from Proposition 8.2 that with probability $1-2\delta$

$$\begin{aligned}\mathrm{II} &\leq \mathcal{R}(h_{S_1}^{SRM}) + \sqrt{\frac{\log(k(h_S^{CV}))}{\alpha m}} + \sqrt{\frac{\log(k(h_{S_1}^{SRM}))}{\alpha m}} + 2\sqrt{\frac{\log(4/\delta)}{2\alpha m}}\\ &\leq \mathcal{R}(h_{S_1}^{SRM}) + 2\sqrt{\frac{\max\{\log(k(h_S^{CV})), \log(k(h_{S_1}^{SRM}))\}}{\alpha m}} + 2\sqrt{\frac{\log(4/\delta)}{2\alpha m}}.\end{aligned}$$

□

If αm is not too small, i.e., when the validation set is large, then we achieve similar results with cross validation to those achieved with structural risk minimisation with sample S_1. However, if this means that S_1 is very small, then, we do not benefit. Hence, the right choice of α is crucial.

9 Lecture 9 - Regression and Regularisation

9.1 Regression and general loss functions

Until now, we have stated most of our results for binary classification problems. In practice, we often have labels that are not necessarily only 0 or 1. Hence, we need to generalise Definition 2.2 and Definition 2.1/ Equation (2).

We do this by invoking the notion of a loss function already introduced in Definition 3.5.

Definition 9.1. *Let L be a loss function on $\mathcal{Y} \times \mathcal{Y}$. Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \mathcal{Y}$ and let $h \in \mathcal{H}$. The* risk of *h is defined by*

$$\mathcal{R}_L(h) = \mathbb{E}_{\mathcal{D}}\left(L(h(x), y)\right).$$

Similarly, we define the empirical risk for a general loss function:

Definition 9.2. *Let L be a loss function on $\mathcal{Y} \times \mathcal{Y}$. Let $h \in \mathcal{H}$, and $S := (x_i, y_i)_{i=1}^m$ be a training sample. The* empirical risk *is defined as*

$$\widehat{\mathcal{R}}_{S,L}(h) = \frac{1}{m} \sum_{i=1}^{m} L(h(x_i), y_i).$$

Note that, Theorem 3.1 yields generalisation bounds for these loss functions.

Example 9.1. *Some loss functions that are quite frequently used:*

- *The 0-1 loss: $L_{0-1}(y_1, y_2) = \mathbb{1}_{y_1 \neq y_2}$. We have used this everywhere until now. Used if $\mathcal{Y} = \{a, b\}$ for $a \neq b$.*
- *The quadratic loss: $L_2(y_1, y_2) = \|y_1 - y_2\|^2$. Used if $\mathcal{Y}$ is a normed space such as $\mathbb{R}^d$, $d \in \mathbb{N}$.*
- *Cross entropy loss/Log Likelihood-loss: $L_{\mathrm{CE}}(y_1, y_2) = -(y_1 \log(y_2) + (1 - y_1)\log(1 - y_2))$. Used if $\mathcal{Y} \subset [0, 1]$, $y_1 \in \{0, 1\}$, $y_2 \in (0, 1)$.*
- *Hinge loss: $L_H(y_1, y_2) = \max\{1 - y_1 y_2\}$. Used if $\mathcal{Y} \subset [-1, 1]$.*

9.2 Linear regression

Using non-binary loss functions, we can now also solve regression problems via empirical risk minimisation. One classical example is linear regression.

In linear regression we have a distribution $\mathcal{D}$ on $\mathbb{R}^p \times \mathbb{R}^q$ and $\mathcal{H}$ is the set of all linear maps from $\mathbb{R}^p \to \mathbb{R}^q$ which we can interpret as $q \times p$ matrices.

Choosing $q = 1$ for simplicity, we have for a sample $S = (x_i, y_i)_{i=1}^m$ and the square loss L_2 that

$$h_{S,L_2}^{ERM}(x) = \langle a, x \rangle, \text{ where}$$
$$\arg\min_{a \in \mathbb{R}^n} \sum_{i=1}^{m} |\langle a, x_i \rangle - y_i|^2.$$

Clearly a is the solution of the least squares problem

$$a = \arg\min_{\overline{a} \in \mathbb{R}^p} \|X\overline{a} - \mathbf{y}\|^2, \tag{30}$$

where the rows of X are the x_i and $\mathbf{y} = (y_i)_{i=1}^m$. One solution of (30) is

$$\hat{a} = (X^T X)^{-1} X^T \mathbf{y}.$$

A small generalisation of linear regression is polynomial regression or more generally basis regression.

Let $(h_k)_{k=1}^K$ be linearly independent such that $\operatorname{span}(h_k)_{k=1}^K = \mathcal{H} \subset \{\mathcal{X} \to \mathbb{R}\}$ for an arbitrary linear space $\mathcal{X}$, then finding

$$\arg\min_{h \in \mathcal{H}} \frac{1}{m} \|h(x_i) - y_i\|^2$$

is equivalent to finding

$$\arg\min_{a \in \mathbb{R}^K} \frac{1}{m} \left| \sum_{k=1}^{K} a_k h_k(x_i) - y_i \right|^2. \tag{31}$$

Hence, setting $X_{i,k} = h_k(x_i)$, we have that

$$\hat{a} = (X^T X)^{-1} X^T \mathbf{y}.$$

solves (31). Finally,

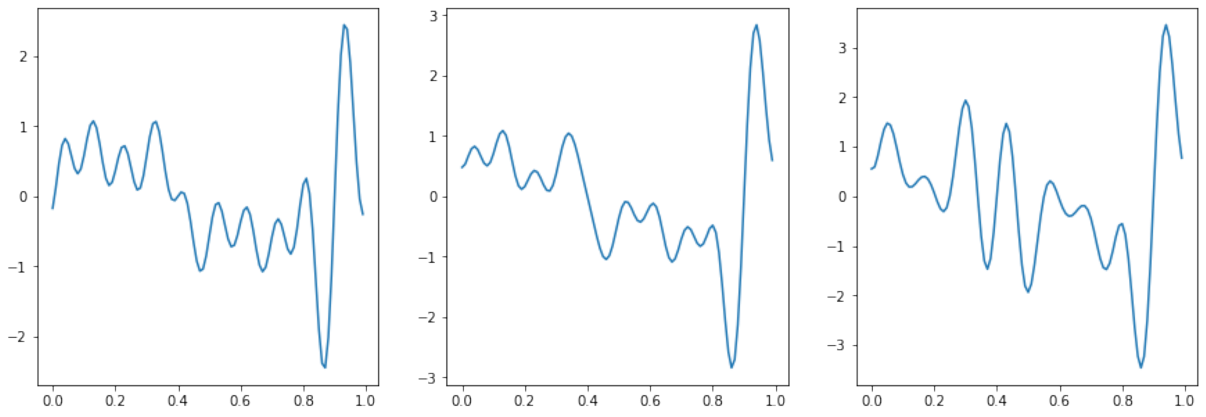
$$h_{S,L_2}^{ERM} = \sum_{k=1}^{K} \hat{a}_k h_k.$$

[2]:
```python
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
```

Let us look at a hypothesis set of sums of sinosoids up to a frequency of 10.

[86]:
```python
x = np.arange(0,1,0.01) # everything lives on [0,1]

sines = np.zeros([100,10])
for k in range(10):
    sines[:,k] = np.sin(2*(k+1)*np.pi*(x + 0.1)) # small shift to not have all start at 0.

plt.figure(figsize = (15, 5))
a = np.random.uniform(0,1, 10)
plt.subplot(1,3,1)
plt.plot(x, np.dot(sines,a))
a = np.random.uniform(0,1, 10)
plt.subplot(1,3,2)
plt.plot(x, np.dot(sines,a))
a = np.random.uniform(0,1, 10)
plt.subplot(1,3,3)
plt.plot(x, np.dot(sines,a))
```

[86]:

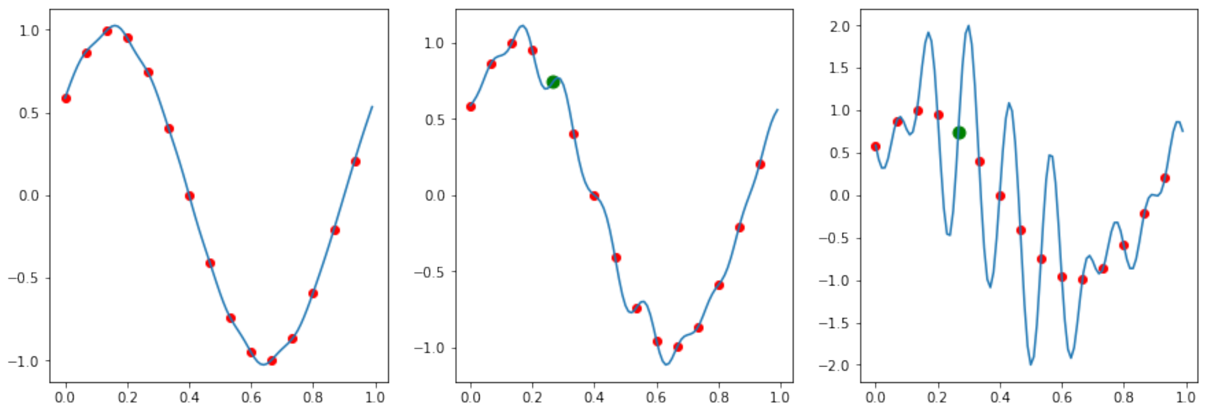


Let us fit some data:

[108]:
```python
num_points = 15

x_dat = np.arange(0,1,1/num_points)
y_dat = np.sin(2*np.pi*(x_dat + 0.1)) # this data should be very easy to fit.

hx_dat = np.zeros([num_points,10])
for k in range(10):
    hx_dat[:, k] = np.sin(2*(k+1)*np.pi*(x_dat + 0.1))



a = np.linalg.lstsq(hx_dat, y_dat, rcond=-1)[0]
```

```
plt.figure(figsize = (15, 5))
plt.subplot(1,3,1)
plt.plot(x, np.dot(sines,a))
plt.scatter(x_dat, y_dat, c = 'r')

# We change one value by almost nothing.
y_dat[4] = y_dat[4] - 1e-15
a = np.linalg.lstsq(hx_dat, y_dat, rcond=-1)[0]
plt.subplot(1,3,2)
plt.plot(x, np.dot(sines,a))
plt.scatter(x_dat, y_dat, c = 'r')
plt.scatter(x_dat[4], y_dat[4], c = 'g',s=80)

# We change one value by ten times almost nothing.
y_dat[4] = y_dat[4] - 1e-14
a = np.linalg.lstsq(hx_dat, y_dat, rcond=-1)[0]
plt.subplot(1,3,3)
plt.plot(x, np.dot(sines,a))
plt.scatter(x_dat, y_dat, c = 'r')
plt.scatter(x_dat[4], y_dat[4], c = 'g',s=80)
```

[108]:



Polynomial/basis regression does not seem to be very stable towards very small changes of a single element. This, however seems to be a quite desirable property, if we want to generalise well. We make this more precise in the next chapter.

9.3 Stability and overfitting

Let $S = (s_k)_{k=1}^m \sim \mathcal{D}^m$ be a sample. We denote by $S^i = (s_k)_{k=1}^m$ the sample with $s_k \sim s_k^i$ for all $k \neq i$ and $s_i^i \sim \mathcal{D}$ independent from S.

Now let $\mathcal{A}$ be a sensible learning algorithm and L be a loss function. Then, we expect that

$$L(\mathcal{A}(S), s_i) \leq L(\mathcal{A}(S^i), s_i),$$

where we use the short-hand notation: $L(\mathcal{A}(S), s_i) = L(\mathcal{A}(S)(x_i), y_i)$, where $s_i = (x_i, y_i)$. On the other hand if

$$L(\mathcal{A}(S), s_i) \ll L(\mathcal{A}(S^i), s_i),$$

then we the algorithm performs only well on the sample s_i if it sees it in its training set. This is a sign of overfitting.

Indeed, we have the following theorem.

Theorem 9.1. *Let $\mathcal{D}$ be a distribution and $S \sim \mathcal{D}^m$. Let $\mathcal{U}$ be the uniform distribution on $[m]$. Further let L be a loss function. Then, we have that for every learning algorithm $\mathcal{A}$,*

$$\mathbb{E}\left(\mathcal{R}_L(\mathcal{A}(S)) - \widehat{\mathcal{R}}_{S,L}(\mathcal{A}(S))\right) = \mathbb{E}\mathbb{E}_{i\sim\mathcal{U}}\left(L(\mathcal{A}(S^i)(x_i), y_i) - L(\mathcal{A}(S)(x_i), y_i)\right), \tag{32}$$

where $(x_i, y_i) = s_i$ and $(x_i^i, y_i^i) = s_i^i$.

Proof. Since (x_i^i, y_i^i) are independent of S, we have that

$$\mathbb{E}\left(\mathcal{R}_L(\mathcal{A}(S))\right) = \mathbb{E}\left(L(\mathcal{A}(S)(x_i^i), y_i^i)\right) = \mathbb{E}\left(L(\mathcal{A}(S^i)(x_i), y_i)\right),$$

where the last equality follows by swapping s_i and s_i^i. Moreover, we have that

$$\mathbb{E}\left(\widehat{\mathcal{R}}_{S,L}(\mathcal{A}(S))\right) = \mathbb{E}\left(\frac{1}{m}\sum_{i=1}^m L(\mathcal{A}(S)(x_i), y_i)\right) = \mathbb{E}\mathbb{E}_{i\sim\mathcal{U}}\left(L(\mathcal{A}(S)(x_i), y_i)\right).$$

The result now follows from the linearity of the expected value. $\square$

Bounding the right hand-side of (32) yields another way to guarantee a small generalisation error. Unfortunately, we have just seen in the previous chapter, that for linear regression we cannot expect the right hand-side of (32) to be small.

We will address this problem in the next chapter. Beforehand, let us fix the the concept of stability used in Theorem 9.1 in form of a definition.

Definition 9.3. *Let $\kappa : \mathbb{N} \to \mathbb{R}$ be monotonically decreasing. A learning algorithm $\mathcal{A}$ is* on-average-replace-one-stable *with rate κ if for every distribution $\mathcal{D}$ every $m \in \mathbb{N}$ it holds that for every sample $S \sim \mathcal{D}^m$:*

$$\mathbb{E}_{S,\mathcal{U}(m)}\left(L(\mathcal{A}(S^i)(x_i), y_i) - L(\mathcal{A}(S)(x_i), y_i)\right) \leq \kappa(m), \tag{33}$$

where $\mathcal{U}(m)$ is the uniform distribution on $[m]$.

9.4 Regularised risk minimisation

We introduce yet another risk minimisation problem. This time we add an auxiliary function to distort the problem in a hopefully sensible way. The effect that we aim at is that we obtain some stability in the sense of Definition (9.3) and Theorem 9.1.

Definition 9.4. *Let $\mathcal{H} = (h_\theta)_{\theta\in\Theta}$ be a hypothesis set, let $L : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ be a loss function. Let S be a sample and $r : \Theta \to \mathbb{R}$. Then we define the solution of* regularised risk minimisation *with* regulariser *r as $h_{S,L}^{RRM} = h_{\theta_{S,L}^{RRM}}$, where*

$$\theta_{S,L}^{RRM} := \arg\min_{\theta\in\Theta} \widehat{\mathcal{R}}_{S,L}(h_\theta, L) + r(\theta).$$

Choosing L to be the 0-1 loss and $\mathcal{H} := \bigcup_{k\in\mathbb{N}} \mathcal{H}_k$ for a sequence of hypothesis sets $(\mathcal{H}_k)_{k\in\mathbb{N}}$ and

$$r(\theta) := \mathfrak{R}_m(\mathcal{H}_{k(h_\theta)}) + \sqrt{\log k(h_\theta)/m}$$

shows that structural risk minimisation is a special case of regularised risk minimisation.

9.5 Tikhonov regularisation

If we have a hypothesis class $\mathcal{H} = (h_\theta)_{\theta\in\Theta}$ where $\Theta \subset \mathbb{R}^d$, then we call the regulariser

$$r_{Tikh,\lambda} \colon \Theta \to \mathbb{R}$$
$$r_{Tikh,\lambda}(\theta) := \lambda\|\theta\|^2$$

Tikhonov regulariser. Here $\|\cdot\|$ is the Euclidean norm. We will see below that this norm can be replaced by any sufficiently convex norm and Θ can be a general normed space.

We are now interested in finding out under which condition regularised risk minimisation with the Tikhonov regulariser admits generalisation bounds. We first study the convexity of $r_{Tikh,\lambda}$.

Definition 9.5. *For a normed space X, we say that a function $f : X \to \mathbb{R}$ is strongly λ-convex if for all $x_1, x_2 \in X$ and all $\alpha \in (0,1)$ it holds that*

$$f(\alpha x_1 + (1-\alpha)x_2) \le \alpha f(x_1) + (1-\alpha) f(x_2) - \frac{\lambda}{2}\alpha(1-\alpha)\|x_1 - x_2\|^2.$$

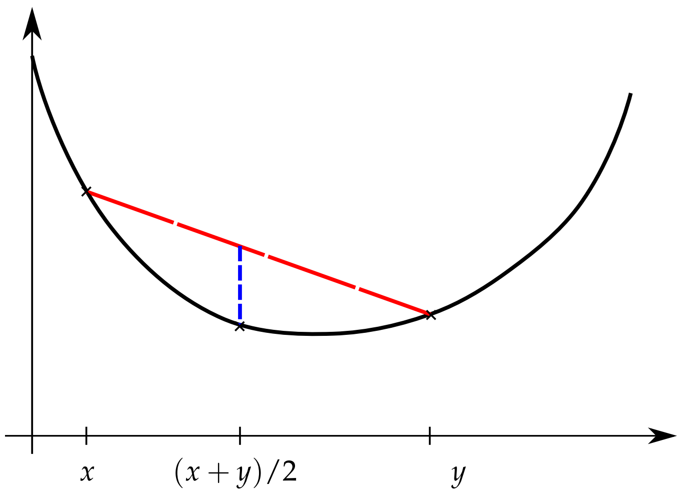


Figure 11: Example of a stronly convex function.

The following lemma will prove useful in the next proof:

Lemma 9.1. *Let X be a normed space and $\lambda > 0$. The following statements hold:*

1. *$r_{Tikh,\lambda}$ is 2λ-strongly convex.*
2. *If $f : X \to \mathbb{R}$ is a λ-strongly convex function and g is a convex function, then $f + g$ is λ-strongly convex.*
3. *If $f : X \to \mathbb{R}$ is a λ-strongly convex and $f(x) = \min_{z\in X} f(z)$, then for every $y \in X$ it holds that*

$$f(y) - f(x) \ge \frac{\lambda}{2}\|x - y\|^2.$$

With these observations in place, we can state the following result:

Proposition 9.1. *Assume that L is a loss function which is ρ-Lipschitz in the first coordinate and is such that $\theta \mapsto L(h_\theta(x), y)$ is convex for every $(x,y) \in \mathcal{X}\times\mathcal{Y}$. Let $S \sim \mathcal{D}^m$ and $\lambda > 0$ and let $\mathcal{A}(S) = h^{RRM}_{S,L}$, where $h^{RRM}_{S,L}$ is the solution of the regularised risk minimisation with regulariser $r_{Tikh,\lambda}$.*
Then, $\mathcal{A}$ is on-average-replacement-one-stable with rate $2\rho^2/(\lambda m)$. In particular,

$$\mathbb{E}\left(\mathcal{R}_L(h^{RRM}_{S,L}) - \widehat{\mathcal{R}}_{S,L}(h^{RRM}_{S,L})\right) \le \frac{2\rho^2}{\lambda m}. \qquad (34)$$

Proof. We write $\mathfrak{R}(\theta; S) = \widehat{\mathcal{R}}_{S,L}(h_\theta)$. Then, by Points 1 and 2 of Lemma 9.1, we conclude that $\theta \mapsto \mathfrak{R}(\theta; S) + r_{Tikh,\lambda}(\theta)$ is 2λ-strongly convex. By Point 3 of Lemma 9.1, we conclude that for every θ' and if θ'' is the minimiser of $\mathfrak{R}(\cdot; S) + r_{Tikh,\lambda}(\cdot)$

$$\mathfrak{R}(\theta'; S) + r_{Tikh,\lambda}(\theta') - (\mathfrak{R}(\theta''; S) + r_{Tikh,\lambda}(\theta'')) \geq \lambda \|\theta'' - \theta'\|^2. \tag{35}$$

For every $i \in [m]$, the left-hand side of (35) can be rewritten as

$$\begin{aligned}
&\mathfrak{R}(\theta'; S) + r_{Tikh,\lambda}(\theta') - (\mathfrak{R}(\theta''; S) + r_{Tikh,\lambda}(\theta'')) \\
&\quad = \mathfrak{R}(\theta'; S^i) + r_{Tikh,\lambda}(\theta') - (\mathfrak{R}(\theta''; S^i) + r_{Tikh,\lambda}(\theta'')) \\
&\qquad + \frac{L(h_{\theta'}(x_i), y_i) - L(h_{\theta'}(x_i^i), y_i^i)}{m} - \frac{L(h_{\theta''}(x_i), y_i) - L(h_{\theta''}(x_i^i), y_i^i)}{m}.
\end{aligned} \tag{36}$$

Now if we choose θ' as the minimiser of $\mathfrak{R}(\cdot; S^i) + r_{Tikh,\lambda}(\cdot)$, then

$$\mathfrak{R}(\theta'; S^i) + r_{Tikh,\lambda}(\theta') - (\mathfrak{R}(\theta''; S^i) + r_{Tikh,\lambda}(\theta'')) \leq 0$$

and hence (36) implies

$$\begin{aligned}
&\mathfrak{R}(\theta'; S) + r_{Tikh,\lambda}(\theta') - (\mathfrak{R}(\theta''; S) + r_{Tikh,\lambda}(\theta'')) \\
&\quad \leq \frac{L(h_{\theta'}(x_i), y_i) - L(h_{\theta'}(x_i^i), y_i^i)}{m} - \frac{L(h_{\theta''}(x_i), y_i) - L(h_{\theta''}(x_i^i), y_i^i)}{m} \qquad (37) \\
&\quad = \frac{L(h_{\theta'}(x_i), y_i) - L(h_{\theta''}(x_i), y_i)}{m} - \frac{L(h_{\theta'}(x_i^i), y_i^i) - L(h_{\theta''}(x_i^i), y_i^i)}{m}. \qquad (38)
\end{aligned}$$

Combined with (35) we now have that

$$\lambda \|\theta'' - \theta'\|^2 \leq \frac{L(h_{\theta'}(x_i), y_i) - L(h_{\theta''}(x_i), y_i)}{m} - \frac{L(h_{\theta'}(x_i^i), y_i^i) - L(h_{\theta''}(x_i^i), y_i^i)}{m}. \tag{39}$$

The estimate of (39) seems to go in the wrong direction to obtain the on-average-replace-one-stability. However, the Lipschitz property of L allows us to perform a boot-strap argument. We observe that

$$|L(h_{\theta'}(x_i), y_i) - L(h_{\theta''}(x_i), y_i)| \leq \rho \|\theta' - \theta''\|. \tag{40}$$

Of course (40) holds when replacing x_i and y_i by x_i^i and y_i^i, respectively. If we plug this equation into (39), we obtain

$$\lambda \|\theta'' - \theta'\|^2 \leq \frac{L(h_{\theta'}(x_i), y_i) - L(h_{\theta'}(x_i^i), y_i^i)}{m} - \frac{L(h_{\theta''}(x_i), y_i) - L(h_{\theta''}(x_i^i), y_i^i)}{m} \leq \frac{2\rho}{m} \|\theta'' - \theta'\|.$$

This implies that

$$\|\theta'' - \theta'\| \leq \frac{2\rho}{\lambda m}.$$

If we combine this estimate with (40), then we conclude that

$$L(h_{\theta'}(x_i), y_i) - L(h_{\theta''}(x_i), y_i) \leq \frac{2\rho^2}{\lambda m}.$$

This implies the on-average-replace-one-stability of $\mathcal{A}$ with rate $\frac{2\rho^2}{\lambda m}$. The "in particular" part of the statement follows from Theorem 9.1. □

Lipschitz continuity of the loss may sometimes be a bit much to ask. For example the very frequently used square loss is not Lipschitz continuous in its input (unless it is restricted to a compact set). The result holds under weaker conditions that include the square loss, too.

Corollary 9.1. *Assume that L is a loss function which is ρ-Lipschitz in the first coordinate and is such that $\theta \mapsto L(h_\theta(x), y)$ is convex for every $x, y \in \mathcal{X} \times \mathcal{Y}$. Let $S \sim \mathcal{D}^m$ and $\lambda > 0$ and let $h_{S,L}^{RRM}$ be the solution of the regularised risk minimisation with regulariser $r_{Tikh,\lambda}$.*
Then the following statements hold:

1. *For all $\theta^* \in \Theta$ it holds that*

$$\mathbb{E}(\mathcal{R}_L(h_{S,L}^{RRM})) \leq \mathcal{R}_L(h_{\theta^*}) + \lambda\|\theta^*\|^2 + \frac{2\rho^2}{\lambda m}. \tag{41}$$

2. *If $\|\theta\| \leq B$ for all $\theta \in \Theta$ and $\lambda = \sqrt{2\rho^2/(B^2 m)}$, then*

$$\mathbb{E}(\mathcal{R}_L(h_{S,L}^{RRM})) \leq \min_{\theta\in\Theta} \mathcal{R}_L(h_\theta) + \rho B\sqrt{\frac{8}{m}}. \tag{42}$$

3. *With probability $1 - B\sqrt{8/(m\epsilon^2)}$ it holds that*

$$\mathcal{R}_L(h_{S,L}^{RRM}) \leq \min_{\theta\in\Theta} \mathcal{R}_L(h_\theta) + \epsilon. \tag{43}$$

Proof. We have from Proposition 9.1 that for every $\theta^* \in \Theta$

$$\begin{aligned}\mathbb{E}\left(\mathcal{R}_L(h_{S,L}^{RRM})\right) &\leq \mathbb{E}\left(\widehat{\mathcal{R}}_{S,L}(h_{S,L}^{RRM})\right) + \frac{2\varrho^2}{\lambda m}\\ &\leq \mathbb{E}\left(\widehat{\mathcal{R}}_{S,L}(h_{\theta^*}) + \lambda\|\theta^*\|^2\right) + \frac{2\varrho^2}{\lambda m}\\ &= \mathcal{R}_L(h_{\theta^*}) + \lambda\|\theta^*\|^2 + \frac{2\varrho^2}{\lambda m}.\end{aligned}$$

where the second inequality follows since the regularised empirical risk is larger than the empirical risk and the fact that $h_{S,L}^{RRM}$ was the minimiser of the regularised risk. This yields (41).

For $\lambda = \sqrt{2\rho^2/(B^2 m)}$, we estimate $\lambda\|\theta^*\|^2$ by $B\sqrt{2\rho^2/m}$ and also have that $\frac{2\varrho^2}{\lambda m} = B\sqrt{2\rho^2/m}$. Applied to (41) this yields (42).

To prove (43), we observe that with (42)

$$\mathbb{E}\left(\mathcal{R}_L(h_{S,L}^{RRM}) - \min_{\theta\in\Theta} \mathcal{R}_L(h_\theta)\right) \leq \rho B\sqrt{\frac{8}{m}}$$

and $\mathcal{R}_L(h_{S,L}^{RRM}) - \min_{\theta\in\Theta} \mathcal{R}_L(h_\theta) \geq 0$. Hence, by Markov's inequality

$$\mathbb{P}\left(\mathcal{R}_L(h_{S,L}^{RRM}) - \min_{\theta\in\Theta} \mathcal{R}_L(h_\theta) > \epsilon\right) \leq \rho B\sqrt{\frac{8}{m\epsilon^2}}.$$

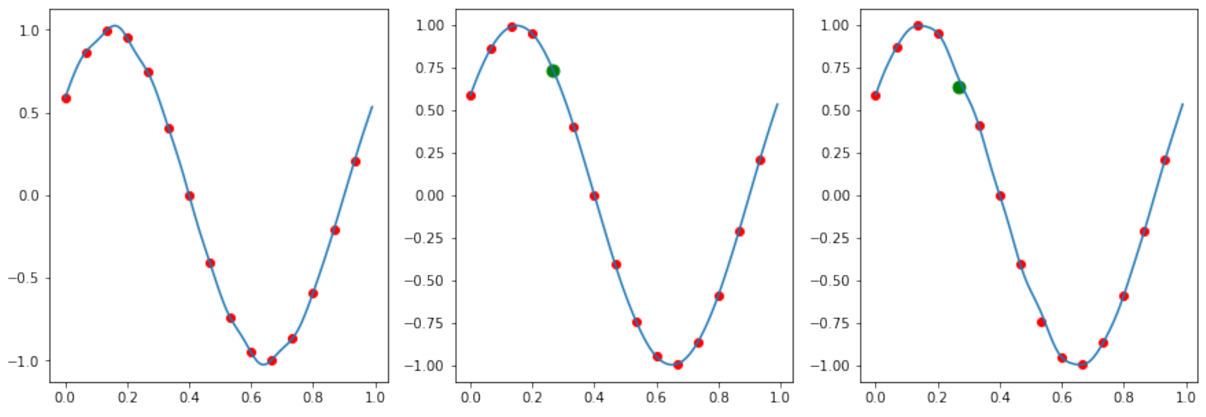
□

Let us end this section by looking at the regularised risk minimisation applied to the regression problem from above:

[134]:
```
num_points = 15
lmbda = 0.0000001
```

```
x_dat = np.arange(0,1,1/num_points)
y_dat = np.sin(2*np.pi*(x_dat + 0.1)) # this data should be very easy to fit.

hx_dat = np.zeros([num_points,10])
for k in range(10):
    hx_dat[:, k] = np.sin(2*(k+1)*np.pi*(x_dat + 0.1))


a = np.linalg.lstsq(hx_dat, y_dat, rcond=-1)[0]

plt.figure(figsize = (15, 5))
plt.subplot(1,3,1)
plt.plot(x, np.dot(sines,a))
plt.scatter(x_dat, y_dat, c = 'r')

# We change one value by something.
y_dat[4] = y_dat[4] - 1e-2
a = np.linalg.lstsq(np.dot(hx_dat.T,hx_dat) + lmbda*np.identity(10), np.dot(hx_dat.T,y_dat), rcond=-1)[0]
plt.subplot(1,3,2)
plt.plot(x, np.dot(sines,a))
plt.scatter(x_dat, y_dat, c = 'r')
plt.scatter(x_dat[4], y_dat[4], c = 'g',s=80)

# We change one value by ten times something.
y_dat[4] = y_dat[4] - 1e-1
a = np.linalg.lstsq(np.dot(hx_dat.T,hx_dat) + lmbda*np.identity(10), np.dot(hx_dat.T,y_dat), rcond=-1)[0]
plt.subplot(1,3,3)
plt.plot(x, np.dot(sines,a))
plt.scatter(x_dat, y_dat, c = 'r')
plt.scatter(x_dat[4], y_dat[4], c = 'g',s=80)
```

```
[134]: <matplotlib.collections.PathCollection at 0x7f62c2c3e668>
```



10 Lecture 10 - Freezing Fritz

Freezing Fritz is a pretty cool guy. He has one problem, though. In his house, it is quite often too cold or to hot during the night. Then he has to get up and open or close his windows or turn the heat up or down. Needless to say, he would like to avoid this.

However, his flat has three doors that he can keep open or closed, it has four radiators, and four windows. There is a picture of his home in Figure 12. It seems like there are endless possibilities of prepping the flat for whatever temperature the night will have.

Fritz, does not want to play his luck any longer and decided to get active. He recorded the temperature outside and inside of his bedroom for the last two years. Now he would like to find a prediction that, given the outside temperature, as well as a certain configuration of his flat, tells him how cold or warm his bedroom will become.

Can you help Freezing Fritz find blissful sleep?

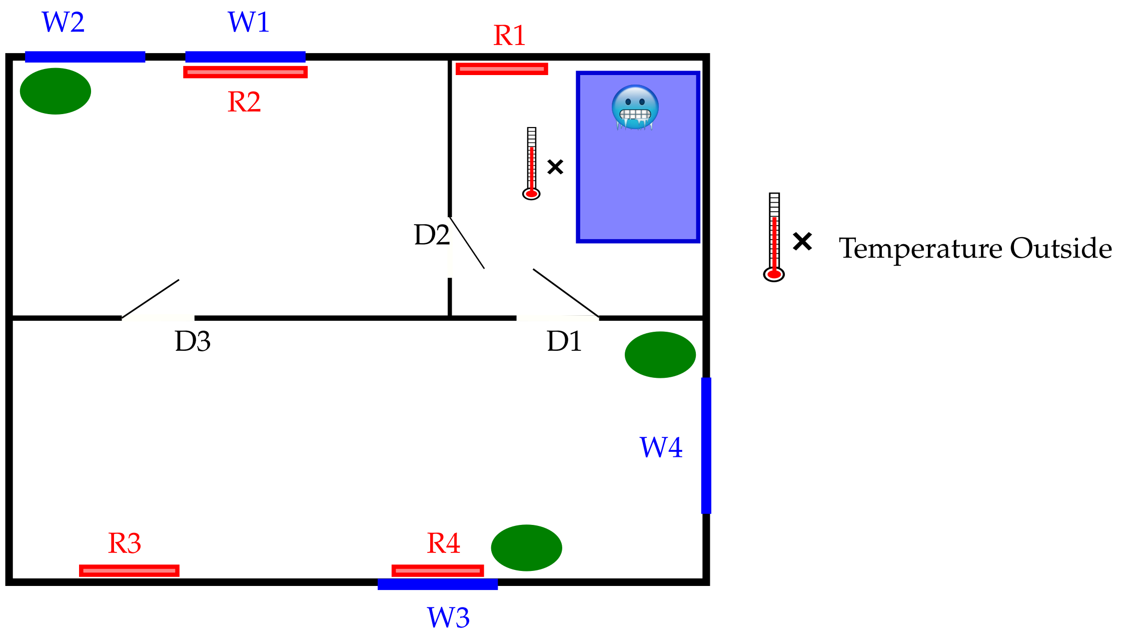


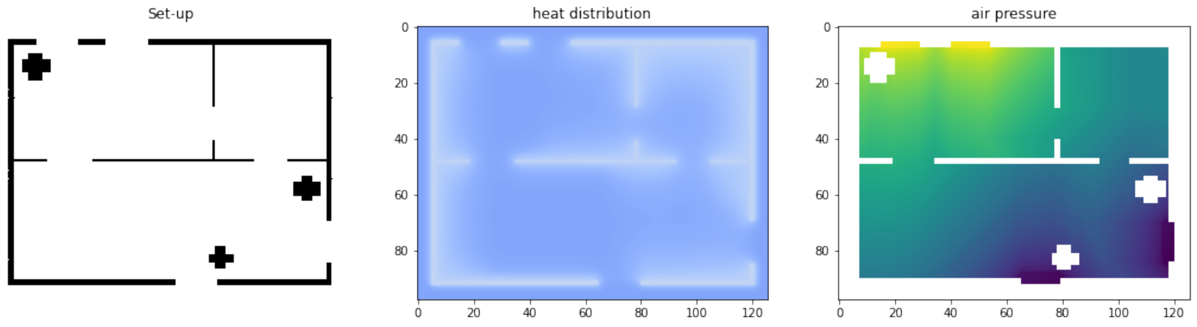
Figure 12: The home of Freezing Fritz. Here we see him lying in his bed. It is too cold. There are four radiators, labelled R1-R4. Four windows labelled W1-W4 and three doors, labelled D1-D3. Fritz also owns three plants. It is unclear, if they have anything todo with the heat distribution in this place, though.

Let us first look at the situation. Below we the experiment that Fritz carried out in 8 cases.

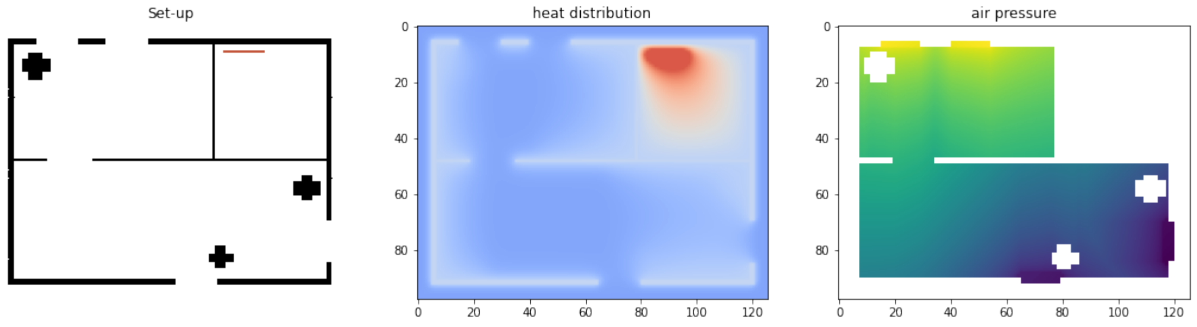
[1]:
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.pyplot import ion
from scipy.signal import convolve2d
import pandas as pd
import seaborn as sn
%matplotlib inline
```

[4]:
```
letItFlow([1,1, 1,1], [0,0,0,0], [1,1,1], 0, report  = True)
letItFlow([1,1, 1,1], [5,0,0,0], [0,0,1], 0, report  = True)
letItFlow([0,0, 0,0], [5,5,5,5], [1,1,1], 0, report  = True)
letItFlow([0,0, 0,0], [0,0,0,0], [1,1,1], 0, report  = True)
letItFlow([1,1, 1,1], [0,0,0,0], [1,1,0], 10, report = True)
letItFlow([1,1, 0,0], [5,5,0,5], [1,1,0], 10, report = True)
letItFlow([0,1, 0,1], [3,5,2,1], [1,0,1], 10, report = True)
letItFlow([0,0, 0,0], [5,5,5,5], [1,1,1], 20, report = True)
```

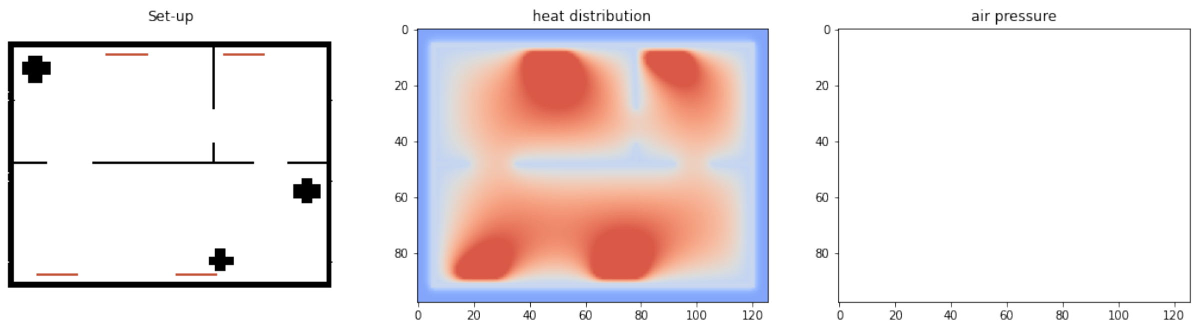
```
/usr/lib/python3/dist-packages/ipykernel_launcher.py:224: RuntimeWarning: divide
by zero encountered in log
```



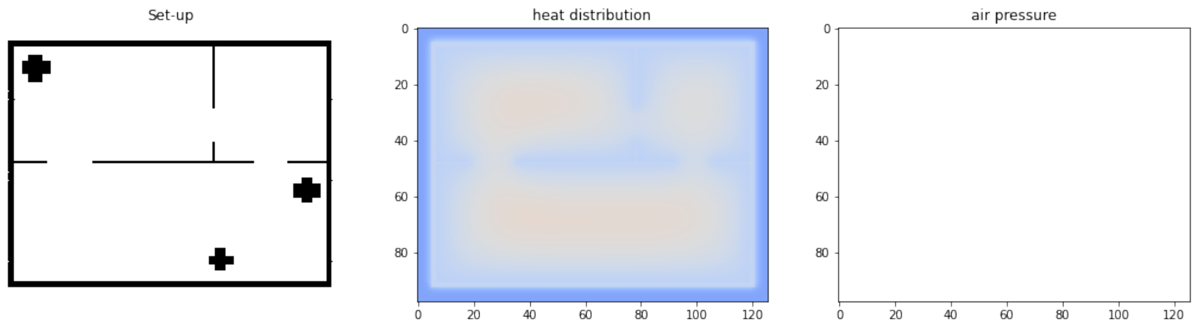
```
Temperature outside: 0°C
Window one open: True, window two open: True, window three open: True, window four open: True
Door one open: True, door two open: True, door three open: True
Heater one level: 0, heater two level: 0, heater three level: 0, heater four level: 0
Temperature in bed: 4.5°C
```



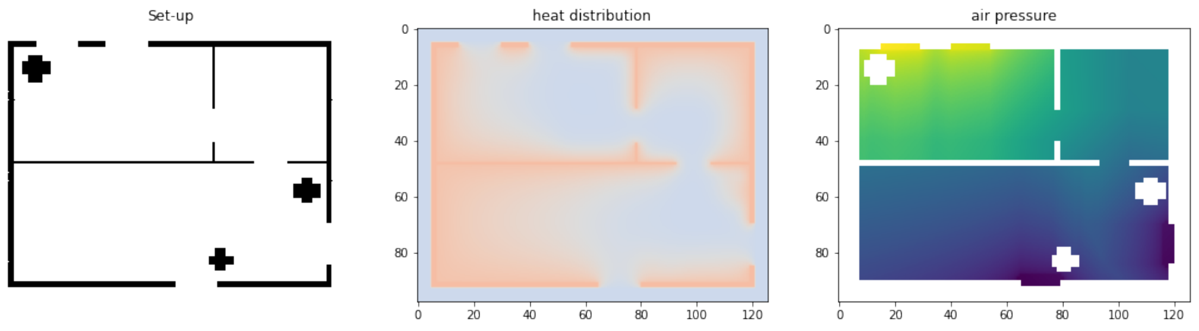
```
Temperature outside: 0°C
Window one open: True, window two open: True, window three open: True, window four open: True
Door one open: False, door two open: False, door three open: True
Heater one level: 5, heater two level: 0, heater three level: 0, heater four level: 0
Temperature in bed: 13.8°C
```



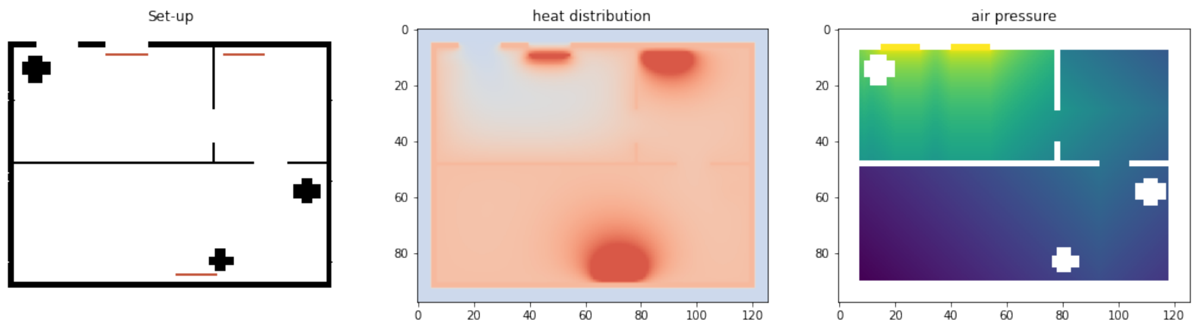
```
Temperature outside: 0°C
Window one open: False, window two open: False, window three open: False, window four open: False
Door one open: True, door two open: True, door three open: True
Heater one level: 5, heater two level: 5, heater three level: 5, heater four level: 5
Temperature in bed: 20.4°C
```



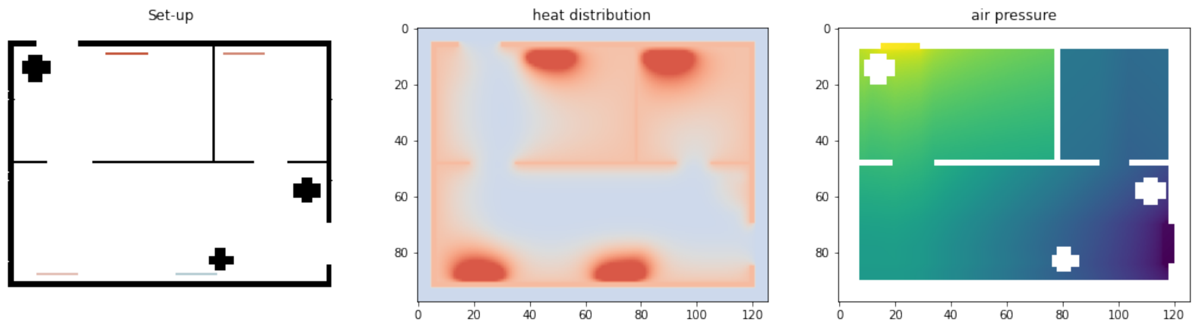
```
Temperature outside: 0°C
Window one open: False, window two open: False, window three open: False, window four open: False
Door one open: True, door two open: True, door three open: True
Heater one level: 0, heater two level: 0, heater three level: 0, heater four level: 0
Temperature in bed: 10.8°C
```



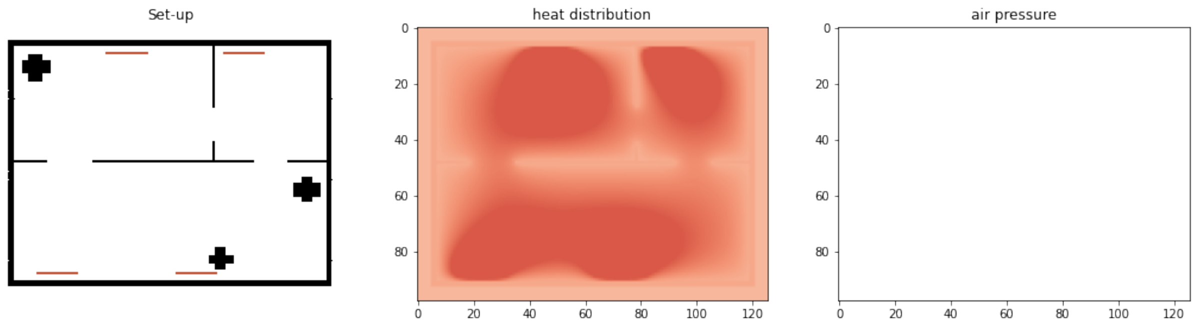
```
Temperature outside: 10°C
Window one open: True, window two open: True, window three open: True, window four open: True
Door one open: True, door two open: True, door three open: False
Heater one level: 0, heater two level: 0, heater three level: 0, heater four level: 0
Temperature in bed: 13.9°C
```



```
Temperature outside: 10°C
Window one open: True, window two open: True, window three open: False, window four open: False
Door one open: True, door two open: True, door three open: False
Heater one level: 5, heater two level: 5, heater three level: 0, heater four level: 5
Temperature in bed: 19.0°C
```



```
Temperature outside: 10°C
Window one open: False, window two open: True, window three open: False, window four open: True
Door one open: True, door two open: False, door three open: True
Heater one level: 3, heater two level: 5, heater three level: 2, heater four level: 1
Temperature in bed: 18.2°C
```



```
Temperature outside: 20°C
Window one open: False, window two open: False, window three open: False, window four open: False
Door one open: True, door two open: True, door three open: True
Heater one level: 5, heater two level: 5, heater three level: 5, heater four level: 5
Temperature in bed: 28.6°C
```

[4]: For the experiment, we first load the data from the data sets 'data_train_Temperature.csv' and 'data_test_Temperature.csv' that will be supplied on moodle.

[8]:
```
data_train_Temperature = pd.read_csv('data_train_Temperature.csv')
data_test_Temperature = pd.read_csv('data_test_Temperature.csv')
data_train_Temperature.head()
```

[8]:
```
   Window 1  Window 2  Window 3  Window 4  Heat Control 1  Heat Control 2  \
0       0.0       0.0       1.0       1.0             5.0             2.0
1       0.0       1.0       1.0       0.0             3.0             3.0
2       0.0       0.0       0.0       1.0             5.0             2.0
3       1.0       1.0       1.0       1.0             1.0             1.0
4       0.0       0.0       1.0       1.0             1.0             0.0

   Heat Control 3  Heat Control 4  Door 1  Door 2  Door 3  \
0             3.0             3.0     1.0     0.0     1.0
1             4.0             0.0     1.0     0.0     1.0
2             2.0             4.0     0.0     0.0     0.0
3             0.0             0.0     1.0     0.0     0.0
4             2.0             2.0     1.0     0.0     0.0

   Temperature Outside  Temperature Bed
0            -3.321343        13.728095
1            -4.474207        15.266521
2            -1.854384        23.918517
3            14.983739        19.458973
4            12.165330        19.635414
```

Let us look at this closely

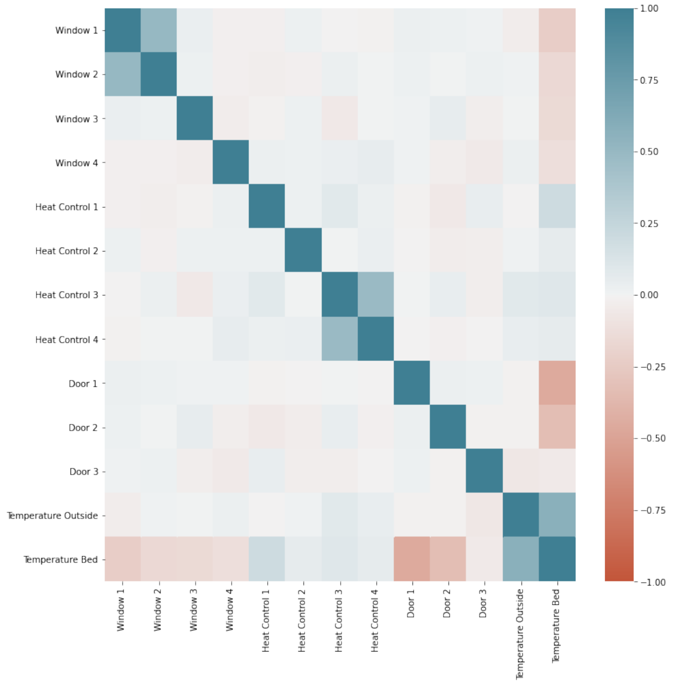
[9]:
```
data_train_Temperature.describe()
```

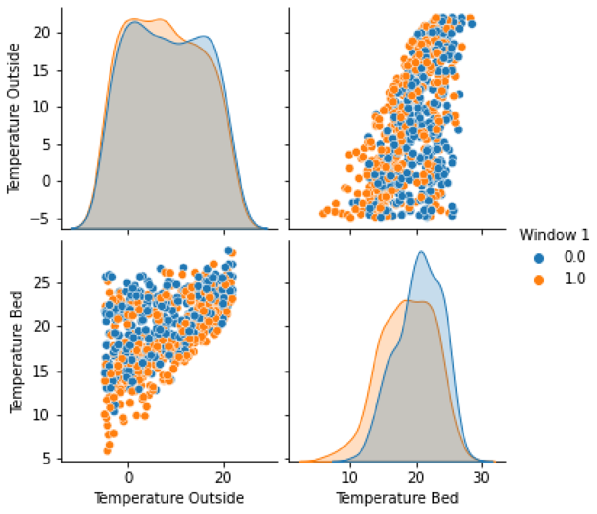
[9]:
```
         Window 1    Window 2    Window 3    Window 4  Heat Control 1  \
count  730.000000  730.000000  730.000000  730.000000      730.000000
mean     0.502740    0.500000    0.509589    0.506849        2.500000
std      0.500335    0.500343    0.500251    0.500296        1.683658
min      0.000000    0.000000    0.000000    0.000000        0.000000
25%      0.000000    0.000000    0.000000    0.000000        1.000000
50%      1.000000    0.500000    1.000000    1.000000        2.000000
75%      1.000000    1.000000    1.000000    1.000000        4.000000
max      1.000000    1.000000    1.000000    1.000000        5.000000

       Heat Control 2  Heat Control 3  Heat Control 4      Door 1      Door 2  \
count      730.000000      730.000000      730.000000  730.000000  730.000000
mean         2.486301        2.420548        2.515068    0.506849    0.495890
std          1.703850        1.703660        1.692529    0.500296    0.500326
min          0.000000        0.000000        0.000000    0.000000    0.000000
25%          1.000000        1.000000        1.000000    0.000000    0.000000
50%          3.000000        2.000000        3.000000    1.000000    0.000000
75%          4.000000        4.000000        4.000000    1.000000    1.000000
max          5.000000        5.000000        5.000000    1.000000    1.000000
```

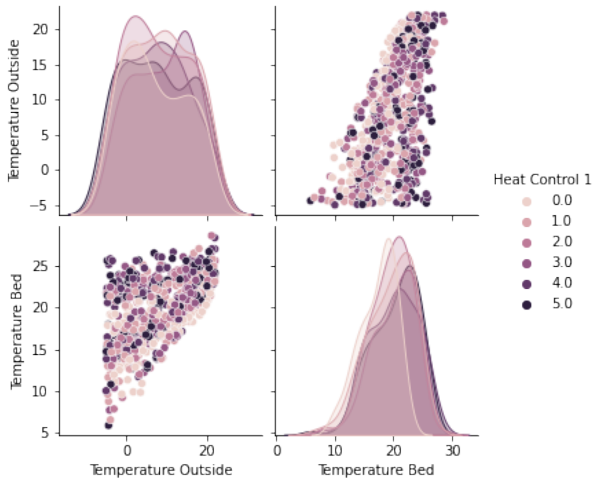
```
        Door 3  Temperature Outside  Temperature Bed
count  730.000000           730.000000       730.000000
mean     0.480822             7.837429        19.530556
std      0.499975             7.788304         3.867791
min      0.000000            -4.998988         5.869975
25%      0.000000             1.039908        16.772720
50%      0.000000             7.470895        20.015297
75%      1.000000            14.628865        22.617748
max      1.000000            21.988839        28.606276
```

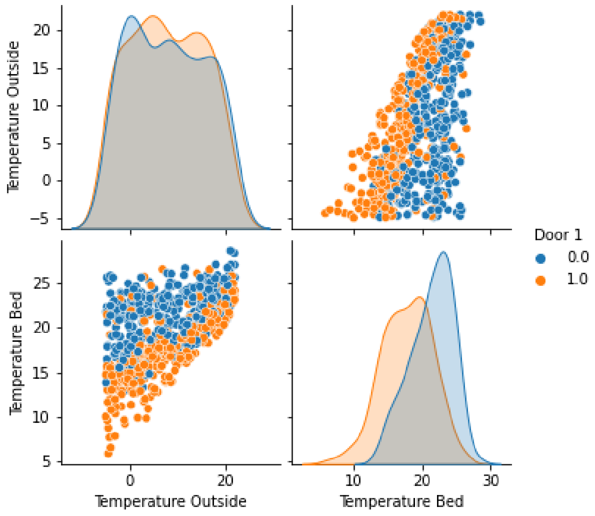
We use the correlation matrix again to see how each of the parameters of the problem affect the temperature in the bedroom. We also look at how the trade-off between outside and inside temperature is affected by some of the parameters.

[10]:
```
corrMatrix = data_train_Temperature.corr()
plt.figure(figsize = (12,12))
palette = sn.diverging_palette(20, 220, n=256)
sn.heatmap(corrMatrix, annot=False, cmap = palette, vmin = -1, vmax = 1)
plt.show()

plt.figure(figsize = (12,12))
sn.pairplot(data_train_Temperature, vars = ['Temperature Outside', 'Temperature Bed'], kind = 'scatter',␣
 ↪hue='Window 1')
sn.pairplot(data_train_Temperature, vars = ['Temperature Outside', 'Temperature Bed'], kind = 'scatter',␣
 ↪hue='Heat Control 1')
sn.pairplot(data_train_Temperature, vars = ['Temperature Outside', 'Temperature Bed'], kind = 'scatter',␣
 ↪hue='Door 1')
plt.show()
```









My idea is to interpolate over the data but weigh it according to my observations and domain knowledge. So I give low weights to windows 2 and 3. Same with door 3. Then I also think that the doors are more important for the overall value than the individual heaters. My predictor is now a simple weighted interpolation over 80% of the data training set. I validate on 20% of the training set.

[32]:
```
observations = data_train_Temperature.shape[0]
simple_data_set = data_train_Temperature.copy().drop(range(int(0.2*observations)), axis = 0)

def predict(data):
    simple_test_set = data.copy()

    #some parameters are more important to fit than others. (In our case, window 1 and doors 1 and 2)
    weights = np.ones(12)
    weights[0]  = 10 # Window 1
    weights[1]  = 0.1 # Window 2
    weights[2]  = 1 # Window 3
    weights[3]  = 10 # Window 4
    weights[4]  = 1 # Heat 1
    weights[5]  = 1 # Heat 2
    weights[6]  = 1 # Heat 3
    weights[7]  = 1 # Heat 4
    weights[8]  = 10 # Door 1
    weights[9]  = 10 # Door 2
    weights[10] = 1 # Door 3
    weights[11] = 1 # Temp Out

    for k in range(simple_test_set.shape[0]):
        value = 0;
        totaldist = 0

        for j in range(simple_data_set.values.shape[0]):
            value = value + simple_data_set.values[j,-1]/(np.linalg.norm(weights*simple_data_set.
↪values[j,:-1] - weights*simple_test_set.values[k,:-1]))**4
            totaldist = totaldist + 1/(np.linalg.norm(weights*simple_data_set.values[j,:-1] -␣
↪weights*simple_test_set.values[k,:-1]))**4
        simple_test_set.values[k, -1] = value/totaldist


    return simple_test_set
```

Now we compute error on validation set:

[33]:
```
validation_set = data_train_Temperature.copy().drop(range(int(0.2*observations), observations), axis = 0)
res = predict(validation_set)
# root mean square error:
np.linalg.norm(res.values[:,-1] - validation_set.values[:,-1])/np.sqrt(validation_set.shape[0])
```

[33]:
```
1.882422855960373
```

The algorithm was not very sophisticated. Nonetheless I came within an accuracy of 2 degrees on the validation set. For me this seems acceptable. Maybe you can help Freezing Fritz even more?

I will just store my prediction on the test set now:

[34]:
```
#make prediction
prediction = predict(data_test_Temperature)
predicted_Temperatures = prediction.values[:,-1]

np.savetxt('PhilippPetersens_Temperature_prediction.csv', predicted_Temperatures, delimiter=',')
```

Please send your result via email to philipp.petersen@univie.ac.at. Your email should include the names of all people who worked on your code, their student identification numbers, a name for your team, and the code used. It should also contain one or two paragraphs of a short description of the method you used.

11 Lecture 11 - Support Vector Machines I

11.1 Definition of the support vector machine algorithm

For $d \in \mathbb{N}$, we consider the hypothesis class of (affine) linear classifiers on $\mathbb{R}^d$:

$$\mathcal{H} = \{x \mapsto \operatorname{sign}(\langle w, x \rangle + b) \colon w \in \mathbb{R}^d, b \in \mathbb{R}\}.$$

This hypothesis class corresponds to a binary classification problem with $\mathcal{Y} = \{-1, 1\}$. We say that a sample $S = (x_i, y_i)_{i=1}^m$ is *linearly separable*, if there exist $w \in \mathbb{R}^d \setminus \{0\}$ and $b \in \mathbb{R}$ such that

$$y_i = \operatorname{sign}(\langle w, x_i \rangle + b) \text{ for all } i \in [m]$$

or equivalently

$$y_i(\langle w, x_i \rangle + b) \geq 0 \text{ for all } i \in [m].$$

Practically, this means that there exists a hyperplane in $\mathbb{R}^d$ splitting $\mathbb{R}^d$ into two parts, where one contains all samples labelled -1 and the other contains all points labelled 1. An illustration is given in Figure 13.

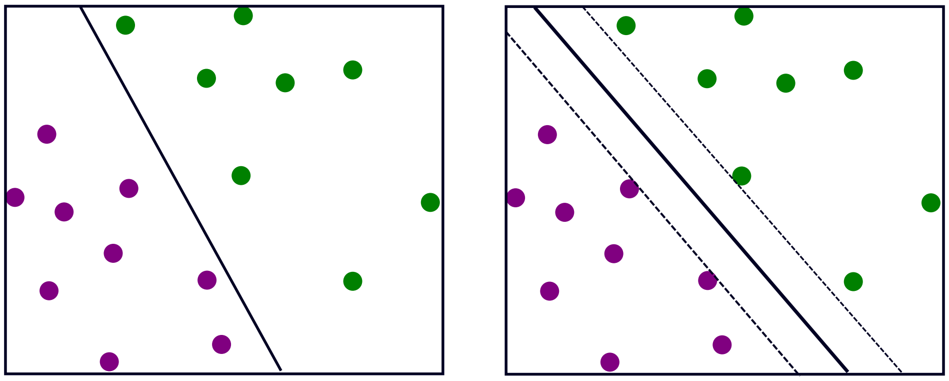


Figure 13: **Left:** Linearly separable data set. **Right:** Hyperplane that maximises the margin.

Definition 11.1. *Let $d \in \mathbb{N}$ and h be a linear classifier. We define the* geometric margin $\rho_h(x)$ *of h at a point z as the Euclidean distance of z to the hyperplane given by $\{h = 0\}$. For a sample $S \in \mathbb{R}^d \times \{-1, 1\}$, we define the* geometric margin *of h as*

$$\rho_h = \min_{i \in [m]} \rho_h(x_i).$$

If $h(x) = \operatorname{sign}(\langle w, x \rangle + b)$, then, for $z \in \mathbb{R}^d$,

$$\rho_h(z) = \frac{|\langle w, z \rangle + b|}{\|w\|_2}. \tag{44}$$

Equation (44) is easily verified by the following argument: Let $H^0 := \{h = 0\}$. For $x_1, x_2 \in H^0$ we have that $\langle w/\|w\|_2, x_1 \rangle + b/\|w\|_2 = 0 = \langle w/\|w\|_2, x_2 \rangle + b/\|w\|_2$ and hence

$$0 = \langle w/\|w\|_2, x_1 - x_2 \rangle = 0.$$

Hence, $w/\|w\|_2$ is a normal on H^0. We conclude that

$$\min_{x \in H^0} \|z - x\|_2 = \min_{x \in H^0} |\langle z, w/\|w\|_2 \rangle - \langle x, w/\|w\|_2 \rangle| = \min_{x \in H^0} |\langle z, w/\|w\|_2 \rangle + b/\|w\|_2| = \frac{|\langle z, w \rangle + b|}{\|w\|_2}.$$

The support vector machine algorithm returns the hyperplane classifier that maximises the geometric margin.

Definition 11.2. *For $d \in \mathbb{N}$, the algorithm $\mathcal{A}_{\mathrm{SVM}}$ takes a sample $S \in (\mathbb{R}^d \times \{-1, 1\})^m$ and outputs a linear classifier $h_{\mathrm{SVM}} = \mathcal{A}_{\mathrm{SVM}}(S)$ such that*

$$\rho_{h_{\mathrm{SVM}}} = \max_{h \in \mathcal{H}} \rho_h.$$

11.2 Primal optimisation problem

We are interested in finding the hyperplane with the maximum geometric margin. By (44), this geometric margin is given by

$$\rho = \max_{w,b\colon y_i(\langle w,x_i\rangle+b)\geq 0} \min_{i\in[m]} \frac{|\langle w,x_i\rangle + b|}{\|w\|}.$$

Since we assume the sample S to be separable, it holds that

$$\max_{w,b\colon y_i(\langle w,x_i\rangle+b)\geq 0} \min_{i\in[m]} \frac{|\langle w,x_i\rangle + b|}{\|w\|} = \max_{w,b} \min_{i\in[m]} \frac{y_i(\langle w,x_i\rangle + b)}{\|w\|} = \max_{\min_{i\in[m]} y_i(\langle w,x_i\rangle+b)=1} \frac{1}{\|w\|}, \tag{45}$$

where the last equality follows by observing that rescaling of (w,b) by any scalar does not affect the value of the fraction.

Since increasing $\|w\|$ will decrease the value (45), we conclude that

$$\rho = \max_{\min_{i\in[m]} y_i(\langle w,x_i\rangle+b)\geq 1} \frac{1}{\|w\|}. \tag{46}$$

Instead of maximising $1/\|w\|$ we can minimise $\frac{1}{2}\|w\|^2$ and we therefore end with the optimisation problem

$$\min_{w,b} \frac{1}{2}\|w\|^2, \tag{47}$$
$$\text{subject to: } y_i(\langle w,x_i\rangle + b) \geq 1. \tag{48}$$

This optimisation problem is strictly convex and the constraints are affine linear. Hence, there exists a unique solution and efficient solvers to find it.

11.3 Support vectors

Now we would like to find a different representation of the solution of the support vector machine algorithm. Therefore, we first need to recall a central result from optimisation theory.

Theorem 11.1 (Karush-Kuhn-Tucker's theorem (simplified)). *Assume that $f\colon \mathcal{X} \to \mathbb{R}$ is convex and differentiable and for $i \in [m]$ let $g_i\colon \mathcal{X} \to \mathbb{R}$ be affine linear. Then x is a solution of the optimisation problem*

$$\min_x f(x)$$
$$\textit{subject to: } g_i(x) \leq 0,$$

if and only if there exists $\alpha \in (\mathbb{R}^+)^m$ such that

$$\nabla_x \mathcal{L}(x,\alpha) = 0 \tag{49}$$
$$(\nabla_\alpha \mathcal{L}(x,\alpha))_i = g_i(x) \leq 0, \quad \textit{for all } i \in [m] \tag{50}$$
$$\sum_{i=1}^m \alpha_i g_i = 0, \tag{51}$$

where

$$\mathcal{L}(x,\lambda) = f(x) + \sum_{i=1}^m \lambda_i g_i \qquad \textit{for } x \in \mathcal{X}, \lambda = (\lambda_1,\dots,\lambda_m) \in (\mathbb{R}^+)^m.$$

Note that, (50) and (51) are equivalent to

$$g_i(x) \le 0 \wedge (\forall i \in [m], \alpha_i g_i(x) = 0). \tag{52}$$

Applying Theorem 11.1 to (46) by defining

$$\mathcal{L}(x,\lambda) = \mathcal{L}(w,b,\lambda) = \frac{1}{2}\|w\|_2^2 + \sum_{i=1}^{m} \lambda_i [y_i(\langle w, x_i\rangle) - 1]$$

yields that for an $\alpha \in (\mathbb{R}^+)^m$

$$\begin{aligned} w &= -\sum_{i=1}^{m} \alpha_i y_i x_i \qquad (53)\\ \sum_{i=1}^{m} \alpha_i y_i &= 0 \\ \alpha_i[y_i(\langle w, x_i\rangle + b) - 1] &= 0, \qquad \forall i \in [m]. \qquad (54) \end{aligned}$$

By the first equality, we conclude that w is a linear combination of the x_i. Moreover, $\alpha_i \neq 0$ only if $y_i(\langle w, x_i\rangle + b) = 1$. We call vectors such that $y_i(\langle w, x_i\rangle + b) = 1$ the *support vectors*. By (45), these are the vectors the distance of which to the hyperplane is equal to the geometric margin.

We notice, that the solution of support vector machine algorithm only depends on the support vectors. This property will be very important in the sequel.

11.4 Generalisation bounds

11.4.1 Leave-one-out analysis

Definition 11.3. *Let $\mathcal{A}$ be a learning algorithm taking samples $S \in (\mathcal{X} \times \{-1,1\})^m$ and mapping them to hypotheses $h\colon \mathcal{X} \to \mathcal{Y}$. The* leave-one-out error *of h on a sample S is defined by*

$$\widehat{R}_{\mathrm{LOO}}(\mathcal{A}, S) := \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}_{h_{S-\{s_i\}}(x_i) \neq y_i},$$

where $S - \{s_i\}$ denotes the sample of size $m-1$ which results from $S = (s_1, \ldots, s_m)$ by removing s_i.

In words, the leave one out error of a sample is the mean error committed when training on all elements of a sample except one and then observing the error on the left out point.

We have that the average leave-one-out error is an unbiased estimator of the risk.

Lemma 11.1. *Let $\mathcal{A}$ be a learning algorithm taking samples $S \in (\mathcal{X} \times \{-1,1\})^m$ and mapping them to hypotheses $h\colon \mathcal{X} \to \mathcal{Y}$. Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \{-1,1\}$. Then it holds that*

$$\mathbb{E}_{S\sim\mathcal{D}^m}(\widehat{R}_{\mathrm{LOO}}(\mathcal{A}, S)) = \mathbb{E}_{S\sim\mathcal{D}^{m-1}}(\mathcal{R}(\mathcal{A}(S)))$$

Proof. From the definition of the leave-one-out error and the linearity of the expected value, we have that

$$\begin{aligned} \mathbb{E}_{S\sim\mathcal{D}^m}(\widehat{R}_{\mathrm{LOO}}(\mathcal{A}, S)) &= \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}_{S\sim\mathcal{D}^m}\left(\mathbb{1}_{h_{S-\{s_i\}}(x_i)\neq y_i}\right) \\ &= \mathbb{E}_{S\sim\mathcal{D}^m}\left(\mathbb{1}_{h_{S-\{s_1\}}(x_1)\neq y_1}\right) \\ &= \mathbb{E}_{S\sim\mathcal{D}^{m-1}}\mathbb{E}_{s\sim\mathcal{D}}\left(\mathbb{1}_{h_S(x)\neq y}\right) \\ &= \mathbb{E}_{S\sim\mathcal{D}^{m-1}}\mathcal{R}(\mathcal{A}(S)). \end{aligned}$$

□

Using this lemma, we can now obtain a first generalisation bound for the SVM algorithm.

Theorem 11.2. *Let, for a linearly separable sample $S \in (\mathcal{X} \times \{-1,1\})^m$, $h_{\mathrm{SVM}} = \mathcal{A}_{\mathrm{SVM}}(S)$ be the hypothesis returned by the SVM algorithm. Let $N_{SV}(S)$ be the number of support vectors that define h_S. Then*

$$\mathbb{E}_{S\sim\mathcal{D}^m}(\mathcal{R}(h_{\mathrm{SVM}})) \leq \mathbb{E}_{S\sim D^{m+1}}\left(\frac{N_{SV}(S)}{m+1}\right). \tag{55}$$

Proof. Let $S = (x_i, y_i)_{i=1}^{m+1}$ be a linearly separable sample of size $m+1$. We observe that if x_i is not a support vector for h_S then by (53) and (54), $h_S = h_{S-\{s_i\}}$. Since S was linearly separable, we have that $h_S(x_i) = y_i$ for all $i \in [m+1]$. We conclude that

$$\widehat{R}_{\mathrm{LOO}}(\mathcal{A}_{\mathrm{SVM}}, S) = \frac{1}{m+1}\sum_{i=1}^{m+1} \mathbb{1}_{h_{S-\{s_i\}}(x_i)\neq y_i} \leq \frac{N_{SV}(S)}{m+1}.$$

The result now follows by Lemma 11.1.

□

12 Lecture 13 - Support Vector Machines II

12.1 Margin theory

Until now we have not seen any benefit from a large geometric margin in the classification by SVMs. Intuitively a large margin makes the classification simpler and should yield improved generalisation bounds.

Defying common sense, one typically starts by studying a different type of margin in a much more general setting first, to ultimately understand the geometric margin in SVM classification. We shall do the same. We will now move from classification by hyperplanes with the 0-1 loss, to classification classification with affine-linear hypotheses and a classification confidence.

If $h(x) = \mathrm{sign}(\langle w, x\rangle + b)$, then

$$\mathbb{1}_{h(x)\neq y_i} = 1 - \mathbb{1}_{(\langle w,x\rangle+b)y_i>0} = 1 - \mathbb{1}_{h(x)y_i>0} \tag{56}$$

Now the right-hand side of (56) makes sense for all real valued functions h. Then, one may interpret the classification with a general $h : \mathcal{X} \to \mathbb{R}$ so that the sign of $h(x)$ corresponds to the predicted class and the magnitude of $h(x)$ corresponds to the confidence of the classification.

From this observation, we can create a loss function that penalises not only wrong classifications but those that do not have enough confidence. In words, for $\rho_c > 0$ we could define the hard margin loss as

$$L_{\mathrm{hard},\rho_c}(y, y') = 1 - \mathbb{1}_{yy'>\rho_c} = \mathbb{1}_{yy'\leq\rho_c}. \tag{57}$$

For analysis purposes it is convenient to consider a continuous alternative of the hard margin loss, which we shall define below:

Definition 12.1. *Let $\rho_c > 0$. The ρ_c-*margin loss *is the function $L_{\rho_c}\colon \mathbb{R}\times\mathbb{R} \to \mathbb{R}^+$ defined as $L_{\rho_c}(y, y') = \Phi_{\rho_c}(yy')$, where $y, y' \in \mathbb{R}$ and*

$$\Phi_{\rho_c}(x) = \min\left\{1, \max\left\{1 - \frac{x}{\rho_c}, 0\right\}\right\} = \begin{cases} 1 & \text{if } x \leq 0 \\ 1 - x/\rho_c & \text{if } 0 \leq x \leq \rho_c \\ 0 & \text{if } \rho_c \leq x. \end{cases}$$

for $x \in \mathbb{R}$

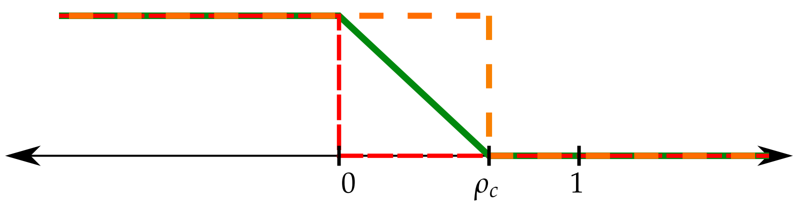


Figure 14: The functions $x \mapsto \mathbb{1}_{x \leq \rho_c}$ (dotted, orange), $\Phi_{\rho_c}(x)$ (solid, green), and $x \mapsto \mathbb{1}_{x \leq 0}$ (dashed, red).

Using the margin loss function, we can define the associated empirical risk function.

Definition 12.2. *For a sample $S = (x_i, y_i)_{i=1}^m$, a margin $\rho_c > 0$, and $h : \mathcal{X} \to \mathbb{R}$, we define the empirical margin loss as*

$$\widehat{R}_{S,\rho_c}(h) := \frac{1}{m} \sum_{i=1}^m \Phi_{\rho_c}(y_i h(x_i)).$$

Note that for all samples S, hypotheses $h\colon \mathcal{X} \to \mathbb{R}$, and margins $\rho_c > 0$

$$\frac{1}{m} \sum_{i=1}^m \mathbb{1}_{y_i h(x_i) \leq 0} \leq \widehat{R}_{S,\rho_c}(h) \leq \frac{1}{m} \sum_{i=1}^m \mathbb{1}_{y_i h(x_i) \leq \rho_c}, \tag{58}$$

see also Figure 14. As mentioned before, the reason to introduce the margin loss function is that it is continuous. More precisely, the fact that it is Lipschitz continuous with Lipschitz constant $1/\rho_c$. The reason why this is beneficial will become clear from the result below:

Theorem 12.1 (Talagrand's Lemma)**.** *Let $\Phi\colon \mathbb{R} \to \mathbb{R}$ be C -Lipschitz for $C > 0$. Then for every hypothesis set $\mathcal{H}$ of real-valued functions it holds that for every sample S*

$$\widehat{\mathfrak{R}}_S(\Phi \circ \mathcal{H}) \leq C \widehat{\mathfrak{R}}_S(\mathcal{H}),$$

where $\Phi \circ \mathcal{H} := \{\Phi \circ h\colon h \in \mathcal{H}\}$.

Now we obtain a first margin based generalisation bound.

Theorem 12.2. *Let $\mathcal{H}$ be a set of real-valued functions. Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \{-1, 1\}$. For $\rho_c > 0$ it holds for all $\delta > 0$ with probability at least $1 - \delta$ over a sample $S \sim \mathcal{D}^m$ and for all $h \in \mathcal{H}$*

$$\mathcal{R}(h) \leq \widehat{R}_{S,\rho_c}(h) + \frac{2}{\rho_c} \mathfrak{R}_m(\mathcal{H}) + \sqrt{\frac{\log(1/\delta)}{2m}} \tag{59}$$

$$\mathcal{R}(h) \leq \widehat{R}_{S,\rho_c}(h) + \frac{2}{\rho_c} \widehat{\mathfrak{R}}_S(\mathcal{H}) + 3\sqrt{\frac{\log(2/\delta)}{2m}}, \tag{60}$$

where $\mathcal{R}(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}}(\mathbb{1}_{yh(x) \leq 0})$.

Proof. Let $\widetilde{\mathcal{H}} := \{(z = (x, y) \mapsto yh(x)\colon h \in \mathcal{H})\}$. Next we set $\widehat{\mathcal{H}} := \Phi_{\rho_c} \circ \widetilde{\mathcal{H}}$ and observe that $\widehat{\mathcal{H}}$ contains functions mapping from $\mathcal{X} \times \{-1, 1\}$ to $[0, 1]$. Thus, we can apply Theorem 3.1, which yields that with probability $1 - \delta$ it holds for all $g \in \widehat{\mathcal{H}}$:

$$\mathbb{E}(g) \leq \frac{1}{m} \sum_{i=1}^m g(z_i) + 2\mathfrak{R}_m(\widehat{\mathcal{H}}) + \sqrt{\frac{\log(1/\delta)}{2m}}. \tag{61}$$

Therefore, for every $h \in \mathcal{H}$

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}(\Phi_{\rho_c}(yh(x))) \leq \widehat{R}_{S,\rho_c}(h) + 2\mathfrak{R}_m(\widehat{\mathcal{H}}) + \sqrt{\frac{\log(1/\delta)}{2m}}. \tag{62}$$

We have that for all $x \in \mathbb{R}$, $\mathbb{1}_{x\leq 0} \leq \Phi_{\rho_c}(x)$ and hence

$$\mathcal{R}(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}}(\mathbb{1}_{y\neq h(x)}) = \mathbb{E}_{(x,y)\sim\mathcal{D}}(\mathbb{1}_{yh(x)\leq 0}) \leq \mathbb{E}_{(x,y)\sim\mathcal{D}}(\Phi_{\rho_c}(yh(x))).$$

We conclude that

$$\mathcal{R}(h) \leq \widehat{R}_{S,\rho_c}(h) + 2\mathfrak{R}_m(\widehat{\mathcal{H}}) + \sqrt{\frac{\log(1/\delta)}{2m}}.$$

Next, we apply Theorem 12.1 which yields that

$$\mathcal{R}(h) \leq \widehat{R}_{S,\rho_c}(h) + \frac{2}{\rho}\mathfrak{R}_m(\widetilde{\mathcal{H}}) + \sqrt{\frac{\log(1/\delta)}{2m}},$$

since Φ_{ρ_c} is $1/\rho_c$-Lipschitz.

Finally, we compute

$$\mathfrak{R}_m(\widetilde{\mathcal{H}}) = \frac{1}{m}\mathbb{E}_S\mathbb{E}_\sigma\left(\sup_{h\in\mathcal{H}}\sum_{i=1}^m \sigma_i y_i h(x_i)\right) = \frac{1}{m}\mathbb{E}_S\mathbb{E}_\sigma\left(\sup_{h\in\mathcal{H}}\sum_{i=1}^m \sigma_i h(x_i)\right) = \mathfrak{R}_m(\mathcal{H}).$$

This yields (59). We can show (60) by following the same steps as above but estimating with the empirical Rademacher complexity in (61). □

As outlined at the beginning of this section, we consider the hypothesis set $\mathcal{H}$ of affine linear functions. Lets compute the empirical Rademacher complexity of this set.

Theorem 12.3. *Let $\mathcal{X}$ be an inner product space and $S \subset B_r(0)$ be a sample of size m. Further let $\mathcal{H} := \{x \mapsto \langle w, x\rangle + b \colon \|w\| \leq \Lambda, |b| \leq s\}$. Then*

$$\widehat{\mathfrak{R}}_S(\mathcal{H}) \leq \sqrt{\frac{(s+r\Lambda)^2}{m}}.$$

Proof. We start by applying the definition of the empirical Rademacher complexity:

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(\mathcal{H}) &= \frac{1}{m}\mathbb{E}_\sigma\left[\sup_{\|w\|\leq\Lambda, |b|\leq\Lambda}\sum_{i=1}^m \sigma_i(\langle w, x_i\rangle + b)\right] \\
&= \frac{1}{m}\mathbb{E}_\sigma\left[\sup_{\|w\|\leq\Lambda, |b|\leq\Lambda}\left\langle w, \sum_{i=1}^m \sigma_i x_i\right\rangle + \sum_{i=1}^m \sigma_i b\right] \\
&\leq \frac{\Lambda}{m}\mathbb{E}_\sigma\left\|\sum_{i=1}^m \sigma_i x_i\right\| + \frac{s}{m}\mathbb{E}_\sigma\left\|\sum_{i=1}^m \sigma_i\right\|.
\end{aligned}$$

We observe that by Jensens inequality and the independence of the σ_i

$$\left(\mathbb{E}_\sigma\|\sum_{i=1}^m \sigma_i\|\right)^2 \leq \mathbb{E}_\sigma\|\sum_{i=1}^m \sigma_i\|^2 = \sum_{i=1}^m \mathbb{E}_\sigma \sigma_i^2 = m.$$

Moreover again by Jensens inequality and the independence of the σ_i, we conclude that

$$\left(\mathbb{E}_\sigma\left[\left\|\sum_{i=1}^m \sigma_i x_i\right\|\right]\right)^2 \leq \mathbb{E}_\sigma\left\|\sum_{i=1}^m \sigma_i x_i\right\|^2 = \sum_{i=1}^m \|x_i\|^2 \leq mr^2.$$

We conclude that

$$\frac{\Lambda}{m}\mathbb{E}_\sigma\left\|\sum_{i=1}^m \sigma_i x_i\right\| + \frac{s}{m}\mathbb{E}_\sigma\left\|\sum_{i=1}^m \sigma_i\right\| \leq \frac{\Lambda r + s}{\sqrt{m}}.$$

□

Now we can combine Theorems 12.3 and 12.2 to obtain the following generalisation bound for affine linear classifiers.

Corollary 12.1. *Let $\mathcal{H} = \{x \mapsto \langle w, x\rangle + b \colon \|w\| \leq \Lambda, |b| \leq s\}$ for $\Lambda, s > 0$. Assume further, that $\mathcal{X}$ is a subset of an inner product space and all elements of $\mathcal{X}$ have norm bounded by r. Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \{-1, 1\}$. Then, $\rho_c > 0$ and for all $\delta > 0$ it holds with probability $1 - \delta$ over a sample $S \sim \mathcal{D}^m$:*

$$\mathcal{R}(h) \leq \widehat{R}_{S,\rho_c}(h) + 2\sqrt{\frac{(s + r\Lambda)^2/\varrho^2}{m}} + \sqrt{\frac{\log(1/\delta)}{2m}},$$

for all $h \in \mathcal{H}$.

Now let $S \subset B_r(0)$ be a linearly separable sample with geometric margin ρ and let $h_S = \mathcal{A}_{SVM}(S)$. Since S is linearly separable, we can assume that the separating hyperplane passes through $B_r(0)$. Let $z \in B_r(0)$ be such that $h_S(z) = 0$, i.e. z lies on the hyperplane.

We have that

$$h_S(x) = \operatorname{sign}(\langle w, x\rangle + b) = \operatorname{sign}(\langle w, x + z\rangle) = h_{S'}(x),$$

where $h_S = \mathcal{A}_{SVM}(S')$ and $S' = (x_i', y_i)_{i=1}^m = (x_i + z, y_i)_{i=1}^m$ results from shifting S by z. Since S was linearly separable with geometric margin ρ we have by (46) that $\|w\| = 1/\rho$. Furthermore, $S' \subset B_{2r}(0)$.

Moreover, by the definition of a geometric margin, we have that

$$\langle w, x_i'\rangle y_i \geq 1$$

for all $i \in [m]$ and hence, $\Phi_1(\langle w, x_i'\rangle y_i) = 0$ for all $i \in [m]$. Corollary 12.1, therefore implies that with probability $1 - \delta$ over the choice of S we have that

$$\mathcal{R}_\mathcal{D}(h_S) = \mathbb{E}_{(x,y)\sim\mathcal{D}}(\mathbb{1}_{h_{S'}(x-z)y \leq 0}) \leq 2\sqrt{\frac{(2r)^2}{m\rho^2}} + \sqrt{\frac{\log(1/\delta)}{2m}}, \tag{63}$$

where we define the right-hand side as ∞ if the sample is not linearly separable. Let us note this as a final corollary.

Corollary 12.2. *For all distributions $\mathcal{D}$ on $\mathcal{X} \times \{-1, 1\}$, where $\mathcal{X}$ is contained in a ball of radius r in a inner product space, it holds that for all $\delta > 0$ with probability $1 - \delta$:*

$$\mathcal{R}_\mathcal{D}(h_S) \leq 4\sqrt{\frac{r^2}{m\rho^2}} + \sqrt{\frac{\log(1/\delta)}{2m}},$$

where ρ is the geometric margin of S.

The estimate of Corollary 12.2 is remarkable in the sense that it does not seem to depend in any way on $\mathcal{X}$. We have seen earlier that the VC dimension of linear classifiers depends on the dimension d. Hence, the VC-dimension-based generalisation bound of Corollary 16.1 would not be dimension independent.

Note though, that we have seen in Theorem 6.1 that the VC dimension based bounds are optimal in the sense that for a carefully designed distribution we cannot significantly outperform the upper bounds. We know now that these bad distributions cannot be such that they only generate linearly separable samples with large geometric margins.

12.2 Non-separable case

Occasionally, one encounters a classification problem that is not linearly separable. One such example, is shown in Figure 15.

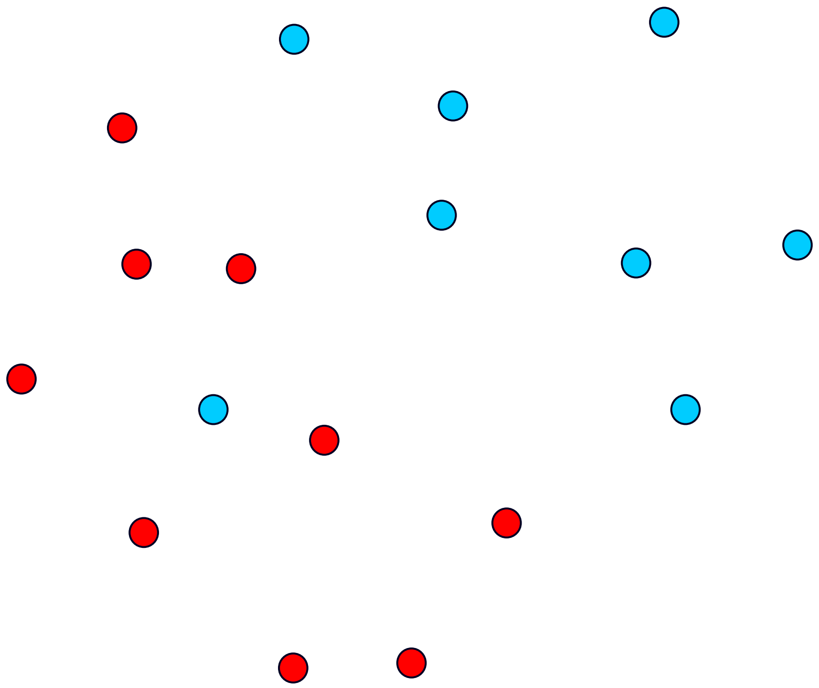


Figure 15: Data set which is not linearly separable.

In this case, no w, b exist such that

$$y_i(\langle w, x_i\rangle + b) \geq 1 \tag{64}$$

for all $(x_i, y_i)_{i=1}^m$. In particular, we cannot define a margin as in (46). Here one typically resorts to a relaxed notion of separating hyperplane, by introducing *slack variables*. Instead of (65), we require that for $(\xi_i)_{i=1}^m$ it holds that

$$y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i. \tag{65}$$

Again, we would like (65) to hold with a large margin $\rho = \frac{1}{\|w\|}$, which in this case, we call *soft-margin*. In addition, we now want the slack variables to be as small as possible. In total, we end up with the optimisation problem:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + \sum_{i=1}^m \xi_i^p$$
$$\text{subject to } y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0,$$

where $p \geq 1$. Similarly to the separable case one can show that the minimiser of the optimisation problem depends only on few sample points. In this case, these points are the support vectors, such that equality holds in (65).

13 Lecture 13 - Freezing Fritz - Discussion

14 Lecture 14 - Kernel methods

Consider the classification problem of Figure 16 below. We see that by tactically increasing the dimension, or more generally speaking by mapping the data to a higher dimensional space, we can sometimes simplify a problem significantly.

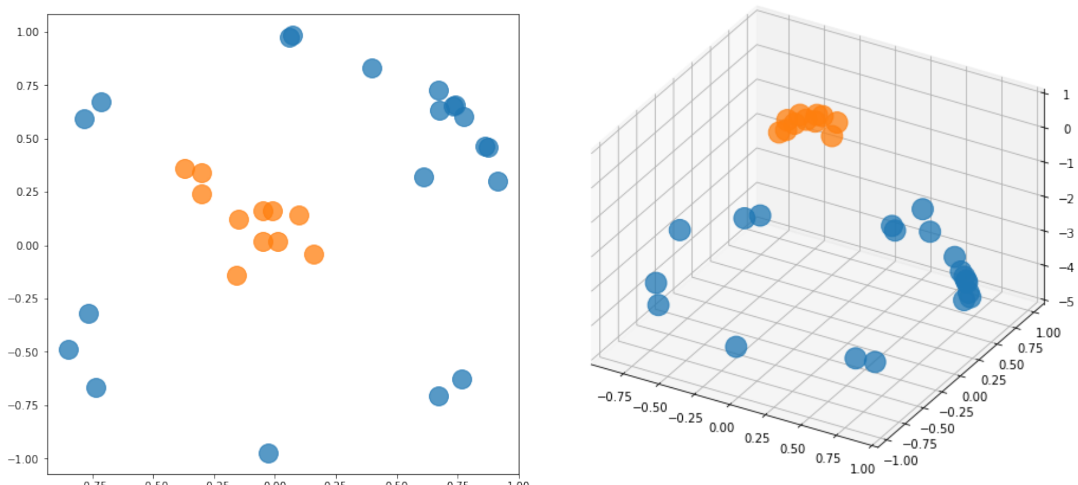


Figure 16: **Left::** Original not linearly separable problem for a sample $S = (x_i, y_i)_{i=1}^{31}$. **Right:** The sample $((x_i, 1 - 5|x_i|), y_i)_{i=1}^{31}$. This seems to be separable.

14.1 The kernel trick and kernel SVM

If we would like to make use of the linear separability after the transformation, then we could use a support vector machine for classification. The overall approach would then look like this:

- Given a domain $\mathcal{X}$ we choose a map $\psi : \mathcal{X} \to \mathcal{Z}$, where $\mathcal{Z}$ is some space.
- For a given sample $S \in (\mathcal{X} \times \mathcal{Y})^m$ we compute the image sample $\widetilde{S} = (\psi(x_i), y_i)_{i=1}^m$.
- Apply the SVM algorithm to $\widetilde{S}$ which yields a linear hypothesis h on $\mathcal{Z}$.
- $h \circ \psi$ is then a classifier for the original problem.

For this procedure to make sense, $\mathcal{Z}$ needs to be such that, we can perform the SVM learning algorithm. This requires $\mathcal{Z}$ to be a space with an inner product. In the sequel, we will therefore assume $\mathcal{Z}$ to be a *Hilbert space*.

It appears to be computationally suboptimal to first embed the data into a high dimensional or even infinite dimensional space and then take high, or even infinite dimensional scalar products. If our algorithms only depend on scalar products between the samples, then it may make sense to use a so-called kernel instead of these scalar products.

Definition 14.1. *Let $\mathcal{X}$ be a set. Then a function $\mathcal{K}\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a* kernel over $\mathcal{X}$.

For a given transformation ψ as above $\mathcal{K}_\psi(x, x') = \langle \psi(x), \psi(x') \rangle_{\mathcal{Z}}$ is a kernel. Often one can, however, also start with a kernel for which then a transform exists.

Theorem 14.1 (Mercer's condition)**.** *Let $\mathcal{X} \subset \mathbb{R}^n$ be compact and let $\mathcal{K} \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a continuous and symmetric function. Then, there exist $\phi_n : \mathcal{X} \to \mathbb{R}$ and $a_n > 0$ such that*

$$K(x, x') = \sum_{n=0}^{\infty} a_n \phi_n(x) \phi_n(x'), \tag{66}$$

if and only if for all $g \in L^2(\mathcal{X})$, the following holds:

$$\int_{\mathcal{X}} \int_{\mathcal{X}} g(x) K(x, x') g(x') dx dx' \geq 0.$$

Theorem 14.1 shows that we do not need to find a specific ψ and an associated Hilbert space as long as we are content with the existence of these objects. We can design a kernel instead. This kernel should satisfy that $\mathcal{K}(x, x')$ is large, if x, x' should be classified similarly.

Having a kernel that can be efficiently computed saves us from computing scalar products between high-dimensional feature embeddings of values $x, x' \in \mathcal{X}$. Nonetheless, to apply the SVM algorithm we need to compute inner products between $\psi(x)$ and a vector w in the Hilbert space. To compute these scalar products efficiently, we make use of the representer theorem.

Consider the following optimisation problem:

$$\min_w f(\langle w, \psi(x_1) \rangle, \dots, \langle w, \psi(x_m) \rangle) + R(\|w\|), \tag{67}$$

where f is an arbitrary function from $\mathbb{R}^m$ to $\mathbb{R}$ and R is a nondecreasing function from $\mathbb{R}^+$ to $\mathbb{R}$. We observe that the SVM algorithm on the feature space is a special case of such an algorithm, when setting $R(\|w\|) = \frac{1}{2}\|w\|^2$ and

$$f(a_1, \dots, a_m) = 1$$

if there exists a $b \in \mathbb{R}$ such that

$$y_i(a_i + b) \geq 1$$

for all $i \in [m]$ and $f(a_1, \dots, a_m) = \infty$ else.

Theorem 14.2. *Assume $\psi : \mathcal{X} \to \mathcal{Z}$, where $\mathcal{Z}$ is a Hilbert space. Assume further that (67) has a solution. Then there exists a vector $\alpha \in \mathbb{R}^N$ such that $w = \sum_{i=1}^N \alpha_i \psi(x_i)$ is a solution of (67).*

Proof. Assume what $w^* \in \mathcal{Z}$. Then, we can write

$$w^* = \bar{w} + u,$$

where $\bar{w} \in \operatorname{span}\{\psi(x_i) \colon i \in [m]\}$ and $u \perp \operatorname{span}\{\psi(x_i) \colon i \in [m]\}$. By the Pythagorean theorem it holds that

$$\|w^*\|^2 = \|\bar{w}\|^2 + \|u\|^2 \geq \|\bar{w}\|^2.$$

Therefore, $R(\|w^*\|) \geq R(\|\bar{w}\|)$. Also

$$f(\langle \bar{w}, \psi(x_1) \rangle, \dots, \langle \bar{w}, \psi(x_m) \rangle) = f(\langle w^*, \psi(x_1) \rangle, \dots, \langle w^*, \psi(x_m) \rangle), \tag{68}$$

by construction. Hence, if there exists an optimal solution of (67) in $\mathcal{Z}$, then there exists also one in $\operatorname{span}\{\psi(x_i) \colon i \in [m]\}$. □

Based on Theorem 14.2, we can now compute for $w = \sum_{i=1}^m \alpha_i \psi(x_i)$

$$\langle w, \psi(x_i)\rangle = \sum_{j=1}^m \alpha_j \langle \psi(x_j), \psi(x_i)\rangle \tag{69}$$

$$\|w\|^2 = \langle w, w\rangle = \sum_{j=1}^m \sum_{i=1}^m \alpha_j \alpha_i \langle \psi(x_j), \psi(x_i)\rangle. \tag{70}$$

If now $\mathcal{K}(x, x') = \langle \psi(x), \psi(x')\rangle$, then the SVM problem on $\mathcal{Z}$ corresponds to the minimisation of

$$\min_{\alpha \in \mathbb{R}^m, b \in \mathbb{R}} \frac{1}{2} \alpha^T G \alpha$$
$$\text{subject to } y_i((G\alpha)_i + b) \geq 1,$$

where $G_{i,j} = \mathcal{K}(x_i, x_j)$ is the so-called *Gram matrix* of the kernel. The optimisation problem above predicts for an new sample x:

$$\operatorname{sign}(\langle w, \psi(x)\rangle + b) = \operatorname{sign}\left(\sum_{i=1}^N \alpha_i \langle \psi(x_i), \psi(x)\rangle + b\right) = \operatorname{sign}\left(\sum_{i=1}^N \alpha_i \mathcal{K}(x_i, x) + b\right).$$

This learning algorithm is called *kernel SVM*.

14.2 Learning guarantees

To obtain learning guarantees of the kernel SVM algorithm, we first compute the Rademacher complexity of the associated hypothesis class.

Proposition 14.1. *Let $\mathcal{K}\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel such that for a function $\psi\colon \mathcal{X} \to \mathcal{Z}$, where $\mathcal{Z}$ is a Hilbert space, it holds that $\mathcal{K}(x, x') = \langle \psi(x), \psi(x')\rangle$ for all $x, x' \in \mathcal{X}$. Let $S = (x_i, y_i)_{i=1}^m \in (\mathcal{X} \times \{-1, 1\})^m$ be a sample so that $\mathcal{K}(x_i, x_i) \leq r^2$ for a given $r > 0$ and all $i \in [m]$.*
We denote $\mathcal{H} := \{x \mapsto \langle w, \psi(x)\rangle + b\colon \|w\|_{\mathcal{Z}} \leq \Lambda, |b| \leq s\}$. Then

$$\widehat{\mathfrak{R}}_S(\mathcal{H}) \leq \frac{\Lambda}{m}\sqrt{Tr(\mathcal{K})} + \frac{s}{\sqrt{m}} \leq \frac{r\Lambda + s}{\sqrt{m}}, \tag{71}$$

where $Tr(\mathcal{K}) = \sum_{i=1}^m \mathcal{K}(x_i, x_i)$ is the trace of $\mathcal{K}$.

Proof. The proof is very similar to that of Theorem 12.3, but we take the effect of the kernel into account. We have per definition that

$$\begin{aligned}\widehat{\mathfrak{R}}_S(\mathcal{H}) &= \frac{1}{m}\mathbb{E}_\sigma\left[\sup_{\|w\| \leq \Lambda, |b| \leq s} \left\langle w, \sum_{i=1}^m \sigma_i \psi(x_i)\right\rangle + \sum_{i=1}^m \sigma_i b\right] \\ &\leq \frac{\Lambda}{m}\mathbb{E}_\sigma\left(\left\|\sum_{i=1}^m \sigma_i \psi(x_i)\right\|\right) + \frac{s}{m}\mathbb{E}_\sigma\left\|\sum_{i=1}^m \sigma_i\right\|.\end{aligned}$$

An application of Jensen's inequality and the independence of σ_i and σ_j for all $i \neq j$ yields

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(\mathcal{H}) &\leq \frac{\Lambda}{m}\left(\mathbb{E}_\sigma\left(\left\|\sum_{i=1}^m \sigma_i \psi(x_i)\right\|^2\right)\right)^{1/2} + \frac{s}{m}\left(\mathbb{E}_\sigma\left\|\sum_{i=1}^m \sigma_i\right\|^2\right)^{1/2} \\
&\leq \frac{\Lambda}{m}\left(\mathbb{E}_\sigma\left(\sum_{i=1}^m \|\psi(x_i)\|^2\right)\right)^{1/2} + \frac{s}{m}\left(\mathbb{E}_\sigma \sum_{i=1}^m \|\sigma_i\|^2\right)^{1/2} \\
&\leq \frac{\Lambda}{m}\left(\mathbb{E}_\sigma\left(\sum_{i=1}^m \mathcal{K}(x_i, x_i)\right)\right)^{1/2} + \frac{s}{\sqrt{m}} \\
&= \frac{\Lambda}{m}\sqrt{Tr(\mathcal{K})} + \frac{s}{\sqrt{m}} \leq \frac{r\Lambda + s}{\sqrt{m}}.
\end{aligned}$$

□

We can apply Theorem 12.2 to Proposition 14.1 which yields the following generalisation bound for large margin kernel SVM classifiers:

Theorem 14.3. *Let $\mathcal{K}\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel such that for a function $\psi\colon \mathcal{X} \to \mathcal{Z}$, where Z is a Hilbert space, it holds that $\mathcal{K}(x, x') = \langle \psi(x), \psi(x')\rangle$ for all $x, x' \in \mathcal{X}$. Let $r^2 := \sup_{x\in\mathcal{X}} \mathcal{K}(x, x)$. We denote $\mathcal{H} := \{x \mapsto \langle w, \psi(x)\rangle + b\colon \|w\|_\mathcal{Z} \leq \Lambda, |b| \leq s\}$. Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \{-1, 1\}$. For $\rho_c > 0$ it holds for all $\delta > 0$ with probability at least $1 - \delta$ over a sample $S \sim \mathcal{D}^m$ and for all $h \in \mathcal{H}$*

$$\mathcal{R}(h) \leq \widehat{R}_{S,\rho_c}(h) + \frac{2(r\Lambda + s)}{\rho_c\sqrt{m}} + \sqrt{\frac{\log(1/\delta)}{2m}} \tag{72}$$

where $\mathcal{R}(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}}(\mathbb{1}_{yh(x)\leq 0})$.

14.3 Some standard kernels

- *Polynomial kernels*: For a constant $c > 0$, the polynomial kernel of degree $d \in \mathbb{N}$ over $\mathbb{R}^N$ is defined by
$$\mathcal{K}(x, x') = (\langle x, x'\rangle + c)^d.$$
For example, if $N = 2$ and $d = 2$ then,
$$\mathcal{K}(x, x') = (x_1 x_1' + x_2 x_2' + c)^2 = \left\langle \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 x_2 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ c \end{bmatrix}, \begin{bmatrix} (x')_1^2 \\ (x')_2^2 \\ \sqrt{2}(x')_1(x')_2 \\ \sqrt{2c}(x')_1 \\ \sqrt{2c}(x')_2 \\ c \end{bmatrix} \right\rangle$$

- *Gaussian/Radial basis function kernel*: For any constant $\sigma > 0$, the Gaussian or RBF kernel over $\mathbb{R}^N$ is defined as
$$\mathcal{K}(x, x') = e^{-\|x-x'\|^2/(2\sigma^2)}$$
It can be shown, see [5], that the Gaussian kernel satisfies the assumptions of Theorem 14.1 and there is a ψ and an *infinite dimensional feature space* $\mathcal{Z}$ such that $\mathcal{K}(x, x') = \langle \psi(x), \psi(x')\rangle_\mathcal{Z}$.

14.4 A numerical example

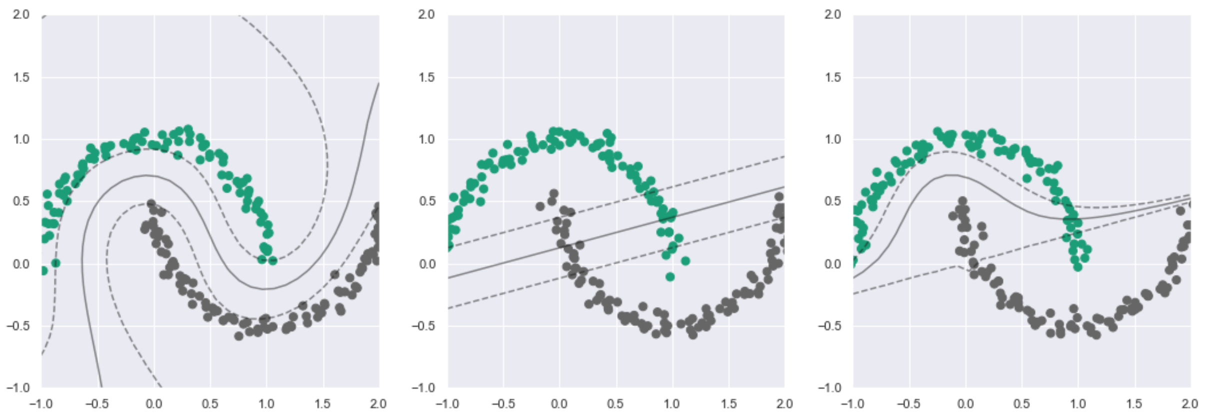
[80]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns; sns.set()
from sklearn.svm import SVC
from sklearn.datasets import make_moons, load_iris, load_wine
```

We use the following helper function to plot the decision regions of our SVM classifiers. This is taken from the Python Data Science Handbook by Jake VanderPlas

[41]:
```python
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""

    # This method is taken from:
    # Python Data Science Handbook
    # by Jake VanderPlas
    # Released November 2016
    # Publisher(s): O'Reilly Media, Inc.
    # ISBN: 9781491912058

    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

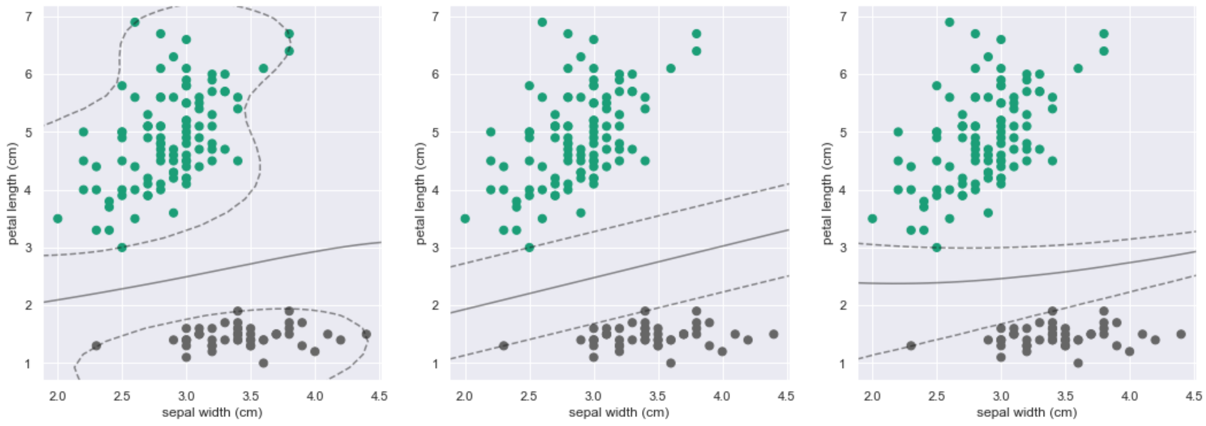
We will now apply the kernel SVM to three different data sets. We start with the moons data set, which consists of two interleaving half-circles.

[109]:
```
def plot_svm_moons(N,  kernel):
    X,y = make_moons(n_samples=N, shuffle=True, noise=0.05)

    model = SVC(kernel=kernel, C=2000)
    model.fit(X, y)

    ax = plt.gca()
    ax.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='Dark2')
    plot_svc_decision_function(model, ax)

N = 200
plt.figure(figsize = (18, 6))
plt.subplot(131)
plot_svm(N, 'rbf')
plt.subplot(132)
plot_svm(N, 'linear')
plt.subplot(133)
plot_svm(N, 'poly')
```



Next, we look at the performance on one of the most famous data sets in data science, the iris data set. This data set contains as features the lengths of two leaves of iris flowers. The associated labels are one of three flower types: "iris setosa', 'iris versicolor', 'iris virginica'. Since our support vector classifier only performs binary classification, we combine iris versicolor, iris virginica to one class.

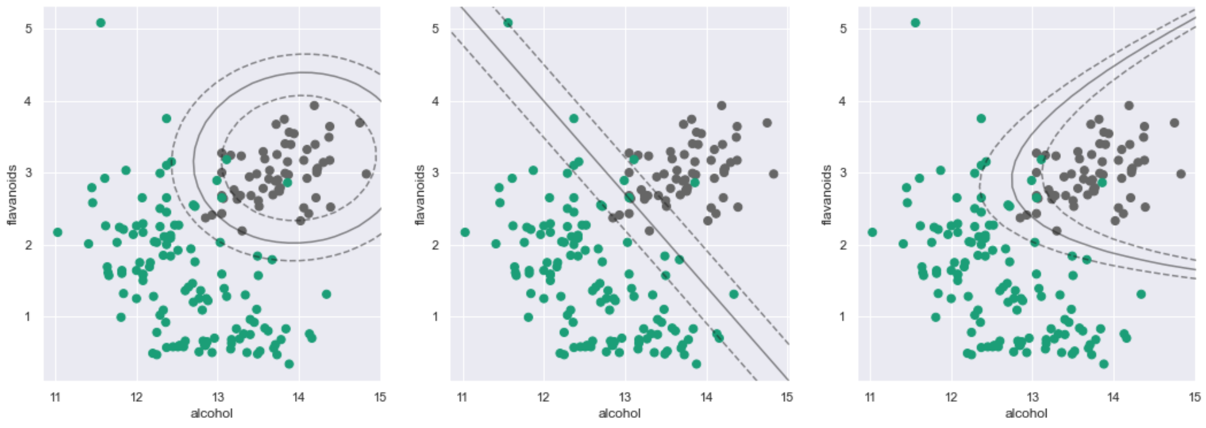
[108]:
```
def plot_svm_iris(N,  kernel):
    X = load_iris().data[:, 1:3]
    y = load_iris().target<1

    X = X[:N, :]
    y = y[:N]

    model = SVC(kernel=kernel, C=2000)
    model.fit(X, y)

    ax = plt.gca()
    ax.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='Dark2')
    ax.set_xlabel(load_iris().feature_names[1])
    ax.set_ylabel(load_iris().feature_names[2])
    plot_svc_decision_function(model, ax)

N = 200
plt.figure(figsize = (18, 6))
plt.subplot(131)
plot_svm_iris(N, 'rbf')
plt.subplot(132)
plot_svm_iris(N, 'linear')
plt.subplot(133)
plot_svm_iris(N, 'poly')
```



Finally, we look at a non linearly separable data set, which is the wine data set. It contains labelled data of three types of wine of which we combine the later two. The data is in the form of specific measurable characteristics of the wine, such as the alcohol or magnesium content.

[111]:
```
def plot_svm_wine(N,  kernel):

    features = [0, 6]
    X = load_wine().data[:, features]
    y = load_wine().target<1

    X = X[:N, :]
    y = y[:N]

    model = SVC(kernel=kernel, C=2000)
    model.fit(X, y)

    ax = plt.gca()
    ax.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='Dark2')
    ax.set_xlabel(load_wine().feature_names[features[0]])
    ax.set_ylabel(load_wine().feature_names[features[1]])
    plot_svc_decision_function(model, ax)
```

```
N = 200
plt.figure(figsize = (18, 6))
plt.subplot(131)
plot_svm_wine(N, 'rbf')
plt.subplot(132)
plot_svm_wine(N, 'linear')
plt.subplot(133)
plot_svm_wine(N, 'poly')
```



15 Lecture 15 - Nearest Neighbour

The nearest neighbour learning algorithm is one of the simplest, but also most versatile and frequently used machine learning algorithms. Let us define it below:

Definition 15.1. *Let $\mathcal{X}$ be a set and $\rho : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a* distance function. *Let $\mathcal{Y}$ be a label space and $S = (x_i, y_i)_{i=1}^m \in (\mathcal{X} \times \mathcal{Y})^m$ be a sample. Let $\pi : \mathcal{X} \to perm(m)$ be such that*

$$\rho(x, x_{\pi(x)(i)}) \leq \rho(x, x_{\pi(x)(i+1)}).$$

Then, we define, for $k \leq m$, the k-nearest neighbour classifier by

$$h_S^{kNN}(x) := \mathcal{A}(S)(x) := \text{ majority label of } (y_{\pi(x)(i)})_{i\in[k]}.$$

Here the majority label is that value y appearing to most in the sequence $(y_{\pi(x)(i)})_{i\in[k]}$.

Of course, we do not have to take the majority label in the definition of $\mathcal{A}(S)$ but could take an average, such as the mean, of the observed labels, if we want to perform regression instead of classification.

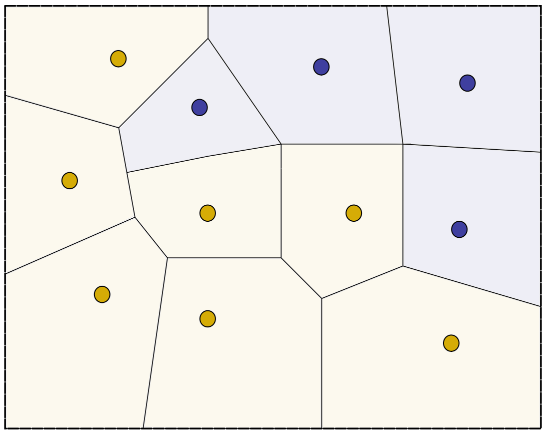


Figure 17: Sketch of a 1-nearest neighbour classifier. The blue and yellow dots are from the training set. The points separate the input space into regions that are closest to one data point. These regions are called Voronoi regions.

15.1 Generalisation bounds for one-nearest-neighbour

Given a distribution on $\mathcal{X}$ and a concept $c : \mathcal{X} \to \mathcal{Y}$, and a sample S, we expect the one-nearest-neighbour rule to work well if the following two conditions are met:

- for most randomly chosen x there exists an x_i in the sample, such that $x - x_i$ is small.
- for x close to x' we have that $c(x)$ is close to $c(x')$.

The second condition is satisfied if c is a Lipschitz function, i.e., if $\|c(x) - c(x')\|_{\mathcal{Y}} \leq \|x - x'\|_{\mathcal{X}}$ for norms on $\mathcal{X}$ and $\mathcal{Y}$. The first condition seems reasonable to assume if just enough samples are already drawn. The following result will be useful.

Lemma 15.1. *Let $Q_1, \dots, Q_r \subset \mathcal{X}$ and let $\mathcal{D}$ be a distribution on $\mathcal{X}$. Further let $S \sim \mathcal{D}^m$. It holds that*

$$\mathbb{E}\left(\sum_{i\colon Q_i \cap S = \emptyset} \mathbb{P}(Q_i)\right) \leq \frac{r}{em}. \tag{73}$$

Proof. By the linearity of the expected value, we have that

$$\mathbb{E}\left(\sum_{i\colon Q_i \cap S = \emptyset} \mathbb{P}(Q_i)\right) = \sum_{i=1}^{r} \mathbb{E}\left(\mathbb{1}_{S \cap Q_i = \emptyset}\right) \mathbb{P}(Q_i).$$

Since $S \sim \mathcal{D}^m$ we have that

$$\mathbb{E}\left(\mathbb{1}_{S \cap Q_i = \emptyset}\right) = \mathbb{P}\left(S \cap Q_i = \emptyset\right) = (1 - \mathbb{P}(Q_i))^m \leq e^{-\mathbb{P}(Q_i)m}.$$

We conclude that

$$\mathbb{E}\left(\sum_{i\colon Q_i \cap S = \emptyset} \mathbb{P}(Q_i)\right) \leq r \max_{i=1,\dots,r} \mathbb{P}(Q_i) e^{-\mathbb{P}(Q_i)m}. \tag{74}$$

The function $h(x) = xe^{-mx}$ satisfies

$$h'(x) = e^{-mx} - mxe^{-mx} = (1 - mx)e^{-mx}$$

and has therefore one root at $x^* = 1/m$. It is not hard to see that this is a maximum of h. Since $h(x^*) = 1/(em)$, we conclude that $h(x) \leq 1/(em)$. Applying this observation to (74), we obtain the result. □

Using the lemma above, we can now prove a generalisation bound for the one-nearest-neighbour classifier, if the underlying concept class is Lipschitz continuous.

Theorem 15.1. *Let $\mathcal{X} = [0,1]^d$, $\mathcal{Y} \subset [0,1]$ and let $\mathcal{D}$ be a distribution on $\mathcal{X}$. Let c be a C_1-Lipschitz continuous target concept and L be a C_2-Lipschitz loss function bounded by 1. It holds that for a sample $S \sim \mathcal{D}^m$ with probability $1-\delta$*

$$\mathbb{E}_S \mathcal{R}_L(h_S^{1NN}) \leq \left(2\sqrt{d}C_1C_2 + \frac{1}{e}\right) m^{-\frac{1}{d+1}}.$$

Proof. For $x \in \mathcal{X}$ and a sample $S \in \mathcal{X}^m$, we denote by $\pi_S(x)$ the closest element to x in S. Then, $h_S^{1NN}(x) = c(\pi(x))$. We have that

$$\begin{aligned}
\mathbb{E}_S \mathcal{R}_L(h_S^{1NN}) &= \mathbb{E}_S \mathbb{E} L(h_S^{1NN}(x), c(x)) \\
&= \mathbb{E}_S \mathbb{E}\left(L\left(h_S^{1NN}(x), c(x)\right) \mathbb{1}_{\|x-\pi(x)\|_\infty \leq \epsilon}\right) + \mathbb{E}_S \mathbb{E}\left(L\left(h_S^{1NN}(x), c(x)\right) \mathbb{1}_{\|x-\pi(x)\|_\infty > \epsilon}\right) \\
&\leq \mathbb{E}_S \mathbb{E}\left(L\left(h_S^{1NN}(x), c(x)\right) \mathbb{1}_{\|x-\pi(x)\|_\infty \leq \epsilon}\right) + \mathbb{E}_S \mathbb{E}\left(\mathbb{1}_{\|x-\pi(x)\|_\infty > \epsilon}\right) =: \mathrm{I} + \mathrm{II}. \qquad (75)
\end{aligned}$$

We start estimating the term I in (75). It holds that

$$\begin{aligned}
L\left(h_S^{1NN}(x), c(x)\right) &= L\left(h_S^{1NN}(x), c(x)\right) - L\left(h_S^{1NN}(x), c(\pi_S(x))\right) \\
&\leq C_2|c(x) - c(\pi_S(x))| \\
&\leq C_1C_2\|x - \pi_S(x)\|,
\end{aligned}$$

due to the Lipschitz regularity of c and L. Moreover, it is not hard to see that

$$\|x - \pi_S(x)\| \leq \sqrt{d}\|x - \pi_S(x)\|_\infty.$$

Hence, $\mathrm{I} \leq C_1C_2\sqrt{d}\epsilon$. To estimate II, we make the following construction. For a given $M \in \mathbb{N}$, we can decompose the domain $\mathcal{X}$ into M^d cubes $Q_1 \ldots Q_{M^d}$ of side-length $1/M$ as in Figure 18. For $2/\epsilon \geq M \geq 1/\epsilon$, we have that if $x_1, x_2 \in Q_i$, then $\|x_1 - x_2\|_\infty \leq \epsilon$. We conclude that $\mathbb{P}(\|x - \pi_S(x)\|_\infty > \epsilon) \leq \mathbb{P}(x \in$

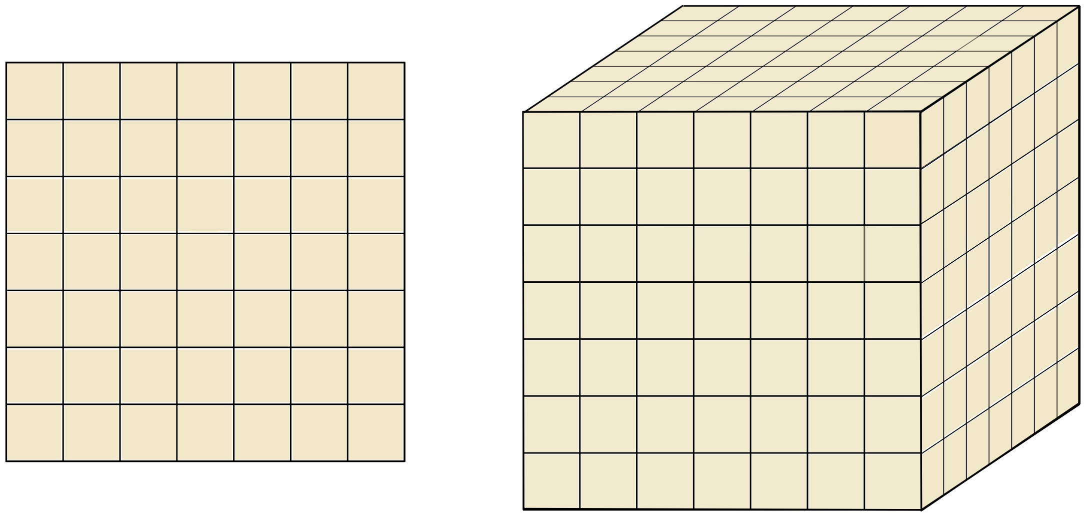


Figure 18: The domain $\mathcal{X}$ can be covered by M^d cubes of side length $1/M$.

$\bigcup_{i:\, Q_i \cap S = \emptyset} Q_i)$. With a union bound and Lemma 15.1, we obtain that

$$\mathrm{II} = \mathbb{E}_S \mathbb{P}(\|x - \pi_S(x)\|_\infty > \epsilon) \leq \frac{M^d}{em} \leq \frac{2^d\epsilon^{-d}}{em}.$$

Choosing $\epsilon = m^{-\frac{1}{d+1}}$ yields that

$$\mathbb{E}_S \mathcal{R}_L(h_S^{1NN}) = \mathrm{I} + \mathrm{II} \leq C_1 C_2 \sqrt{d} m^{-\frac{1}{d+1}} + m^{\frac{d}{d+1}}/em \leq \left(C_1 C_2 \sqrt{d} + e\right) m^{-\frac{1}{d+1}},$$

which yields the claim. □

Remark 15.1. *Note that the generalisation bound of 1-nearest neighbour classification deteriorates exponentially fast with increasing dimension. This is one instance of the so-called* curse of dimension.

16 Lecture 16 - Neural Networks

We start by directly defining neural networks.

Definition 16.1. *Let $N, d \in \mathbb{N}$, $\varrho : \mathbb{R} \to \mathbb{R}$. A* (shallow) neural network *is a function Φ of the form*

$$\Phi \colon \mathbb{R}^d \to \mathbb{R}$$
$$\Phi(x) = \sum_{i=1}^{N} c_i \varrho(\langle a_i, x\rangle + b_i) + e,$$

where $c_i, b_i, d \in \mathbb{R}$ and $a_i \in \mathbb{R}^d$ for $i \in [N]$. We say that
- *Φ has* input dimension *d,*
- *Φ has N* neurons*,*
- *the* activation function *of Φ is ϱ,*
- *the a_i, c_i for $i \in [N]$ are the* weights *of the neural network*
- *the b_i, e are the* biases *of the neural network.*

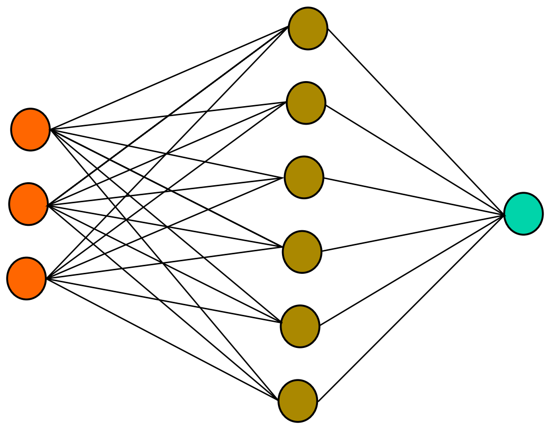


Figure 19: Sketch of a neural network with input dimension 3 and 6 neurons.

In practice, often *deep neural networks* are used. These are functions that result from stacking multiple of these neural networks after another in multiple layers. We will not discuss these types of networks here.

Neural networks form a general class of hypothesis set. If $\rho = 2\mathbb{1}_{(0,\infty)} - 1$, $N = 1$, $d = 0$, and $c_1 = 1$, then the class of such neural networks is the class of hyperplane classifiers that we have already encountered in Section 11.

We first would like to understand this set a bit better and in particular the role of the number of neurons and the activation function. The following result is one of the most famous in neural network theory:

Theorem 16.1 (Universal approximation theorem). *Let $\varrho : \mathbb{R} \to \mathbb{R}$ be a sigmoidal function, i.e., ϱ is continuous and $\lim_{x\to-\infty} \varrho(x) = 0$ and $\lim_{x\to\infty} \varrho(x) = 1$. Then, for every compact set $K \subset \mathbb{R}^d$ and every continuous function $f : K \to \mathbb{R}$ and every $\epsilon > 0$, there exists Φ such that*

$$\sup_{x\in K} |f(x) - \Phi(x)| < \epsilon, \tag{76}$$

where Φ is a neural network with activation function ϱ.

Multiple proofs of this statement or generalisations thereof have been found in the literature, see [3, 4, 2]. We present a proof that is close to that in [2].

Proof. Assume towards a contradiction that there exists a function $f : K \to \mathbb{R}$ and an $\epsilon > 0$ such that for all neural networks Φ with activation function ϱ

$$\sup_{x\in K} |f(x) - \Phi(x)| > \epsilon.$$

Let us denote the set of all neural networks with input dimension d and activation function ϱ by $\mathcal{NN}_{d,\varrho}$. It is clear from the definition that $\mathcal{NN}_{d,\varrho}$ is a subspace of the space of continuous functions on K, which we denote by $C(K)$.

By the theorem of Hahn-Banach, there exists a continuous linear functional $h \in C(K)'$, the dual space of $C(K)$, such that

$$h(\Phi) = 0 \text{ and } h(f) = 1,$$

for all $\Phi \in \mathcal{NN}_{d,\varrho}$. Furthermore, by the representation theorem of Riesz, there exists a signed Borel measure $\mu \neq 0$ such that

$$h(g) = \int_K g(x) d\mu(x).$$

Since $h(\Phi) = 0$ for all neural networks Φ, it holds in particular for every neural network with one neuron:

$$x \mapsto \varrho(\langle a, x\rangle + b).$$

Hence, we conclude that for a non-zero measure μ

$$\int_K \varrho(\langle a, x\rangle + b) d\mu(x) = 0$$

for all $a \in \mathbb{R}^d$ and $b \in \mathbb{R}$.

Since ϱ is continuous and tends to 0 or 1 for $x \to \pm\infty$ respectively, we conclude that ϱ is bounded and for $\lambda, \mu > 0$

$$\varrho(\langle \lambda a, x\rangle + \lambda b + \mu) = \varrho(\lambda(a, x\rangle + b) + \mu) \to \begin{cases} 1 & \text{if } (a, x\rangle + b) > 0, \\ 0 & \text{if } (a, x\rangle + b) < 0, \\ \varrho(\mu) & \text{if } (a, x\rangle + b) = 0. \end{cases}$$

for $\lambda \to \infty$. Letting also $\mu \to \infty$ we see that for every $x \in \mathbb{R}$

$$\varrho(\langle \lambda a, x\rangle + \lambda b + \mu) \to \mathbb{1}_{[0,\infty)}(\lambda a, x\rangle).$$

We conclude by the dominated convergence theorem that for all $a \in \mathbb{R}^d$ and $b \in \mathbb{R}$

$$\int_K \mathbb{1}_{[b,\infty)}(\langle a, x\rangle) d\mu(x) = \int_K \mathbb{1}_{[0,\infty)}(\langle a, x\rangle - b) d\mu(x) = 0.$$

By using the linearity of the integral, we conclude that for all $a \in \mathbb{R}^d$ and $b_1, b_2 \in \mathbb{R}$

$$\int_K \mathbb{1}_{[b_1,b_2)}(\langle a, x\rangle)\mu(x) = 0.$$

Since every univariate continuous function on an interval can be approximated arbitrarily well uniformly by step functions we conclude that for every $g \in C(\mathbb{R})$

$$\int_K g(\langle a, x\rangle)\mu(x) = \int_K g_{[c_1,c_2]}(\langle a, x\rangle)\mu(x) = 0,$$

where $c_1 = \min\{\langle a, x\rangle : x \in K\}$, $c_2 = \max\{\langle a, x\rangle : x \in K\}$.

In particular, $g = \sin$ and $\cos$ is possible and by Euler's formula $e^{ix} = i\sin(x) + \cos(x)$, we conclude that

$$\int_K e^{i(\langle a,x\rangle)}\mu(x) = \int e^{i(\langle a,x\rangle)}\tilde{\mu}(x),$$

for a measure $\tilde{\mu}$ on $\mathbb{R}^d$ supported on K that coincides with μ on K. We conclude that the Fourier transform of $\tilde{\mu}$ vanishes. This implies that $\tilde{\mu}$ and hence μ vanishes, which is a contradiction to the choice of μ. □

We see that neural networks are a versatile hypothesis set, since they can represent every continuous function arbitrarily well, if they are sufficiently large. From a generalisation point of view this is of course not so exciting since the VC dimension of the set of continuous functions is infinite. We recall from Theorem 6.1 that an infinite VC dimension prohibits us from learning anything.

In practice, neural networks with only a finite number of neurons are used. Typically the resulting set of neural networks does then not yield form dense subset of the set of continuous functions. Then, we have again a chance to learn something. Indeed, we can bound the VC dimension of sets of neural networks. The definition of VC dimension requires a function class with outputs in $Y = \{-1, 1\}$. Thus, we can only define a VC dimension for the set of NNs with binary output, which we get by composing every NN with a sign function.

Theorem 16.2 ([1, Theorem 2.1]). *Let $d, N \in \mathbb{N}$ and let ϱ be a piecewise polynomial function. We denote the set of neural networks with N neurons input dimension d and activation function ϱ by $\mathcal{F}_N$. It holds that*

$$VCdim(\operatorname{sign} \circ \mathcal{F}) = \mathcal{O}(N \log N), \text{ for } N \to \infty.$$

We can combine this theorem with Corollary 16.1 which yields the following generalisation bound:

Corollary 16.1. *Let $d, N \in \mathbb{N}$ and let ϱ be a piecewise polynomial function. We denote the set of neural networks with N neurons input dimension d and activation function ϱ by $\mathcal{F}_N$. Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \mathcal{Y}$ with $\mathcal{Y} = \{-1, 1\}$ and $S \sim \mathcal{D}^m$. Then, for every $\delta > 0$, with probability at least $1 - \delta$ for any $h \in \mathcal{F}$:*

$$|\mathcal{R}(\operatorname{sign}(h)) - \widehat{\mathcal{R}}_S(\operatorname{sign}(h))| = \mathcal{O}\left(\sqrt{\frac{2N\log(N)\log(\frac{em}{N\log(N)})}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}\right),$$

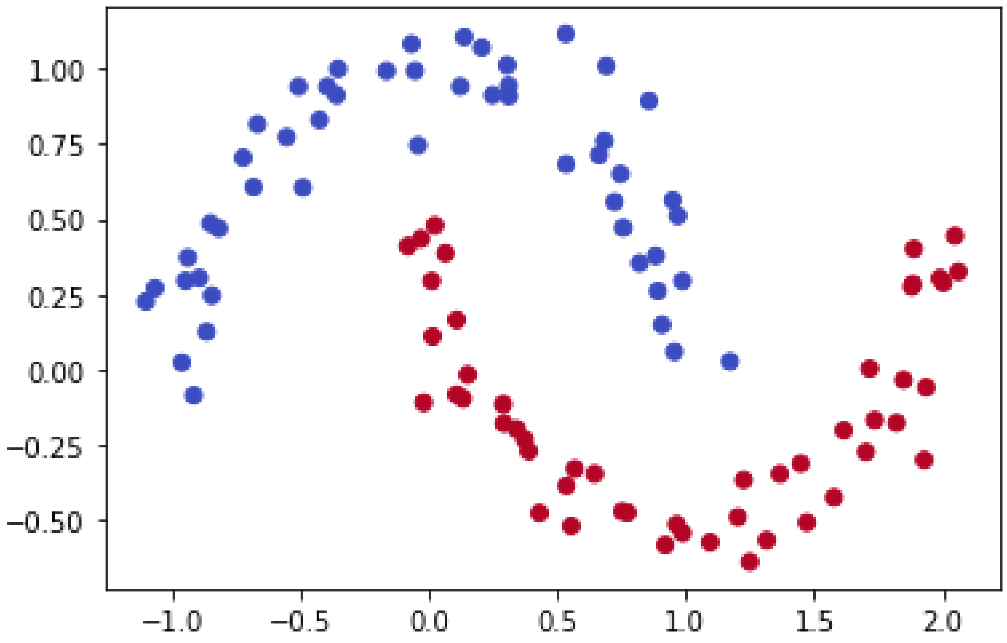
where $m \geq N \log N$ and $N \to \infty$.

[166]:
```python
import numpy as np
import keras as ks
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
```

We test a neural network on the classical moons data set

[175]:
```
X, y = make_moons(noise=0.1, random_state=0)
plt.scatter(X[:,0], X[:,1], c = y, cmap = 'coolwarm')
```

[175]:
```
<matplotlib.collections.PathCollection at 0x7fad8efd2908>
```



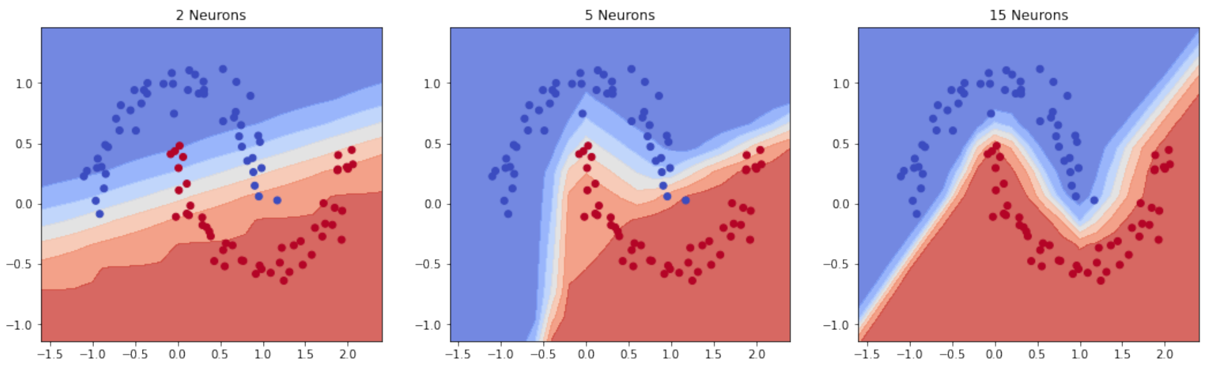
[222]:
```
# We define three models. Always one hidden layer with the relu (x \mapsto \max \{x, 0\}) activation␣
↪function.
# The first model has 2 neurons, the second 5, and the last has 20 neurons.
# We apply a sigmoid to the output for for stability reasons.
model1 = Sequential()
model1.add(Dense(2, input_dim=2, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))

model2 = Sequential()
model2.add(Dense(5, input_dim=2, activation='relu'))
model2.add(Dense(1, activation='sigmoid'))

model3 = Sequential()
model3.add(Dense(20, input_dim=2, activation='relu'))
model3.add(Dense(1, activation='sigmoid'))

# compile the models. Here we need to chose an optimiser. This one is called adam, it is used to
# determine how the training is performed. We do not care in this lecture how this is done.
opt = ks.optimizers.Adam(learning_rate=0.02)
model1.compile(loss='mean_squared_error', optimizer=opt,  metrics=['accuracy'])
model2.compile(loss='mean_squared_error', optimizer=opt,  metrics=['accuracy'])
model3.compile(loss='mean_squared_error', optimizer=opt,  metrics=['accuracy'])
```

```
# fit the models on the dataset

model1.fit(X, y, epochs=30, batch_size=5, verbose = False)
model2.fit(X, y, epochs=30, batch_size=5, verbose = False)
model3.fit(X, y, epochs=30, batch_size=5, verbose = False)

# evaluate the keras model
print('Model 1:')
_, accuracy1 = model1.evaluate(X, y)
print('Model 2:')
_, accuracy2 = model2.evaluate(X, y)
print('Model 3:')
_, accuracy3 = model3.evaluate(X, y)



h=0.2
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

Z1 = model1.predict(np.c_[xx.ravel(), yy.ravel()])
Z1 = Z1.reshape(xx.shape)
Z2 = model2.predict(np.c_[xx.ravel(), yy.ravel()])
Z2 = Z2.reshape(xx.shape)
Z3 = model3.predict(np.c_[xx.ravel(), yy.ravel()])
Z3 = Z3.reshape(xx.shape)

plt.figure(figsize = (18,5))

plt.subplot(1,3,1)
plt.contourf(xx, yy, Z1, cmap='coolwarm', alpha=.8)
plt.scatter(X[:,0], X[:,1], c = y, cmap = 'coolwarm')
plt.title('2 Neurons')

plt.subplot(1,3,2)
plt.contourf(xx, yy, Z2, cmap='coolwarm', alpha=.8)
plt.scatter(X[:,0], X[:,1], c = y, cmap = 'coolwarm')
plt.title('5 Neurons')

plt.subplot(1,3,3)
plt.contourf(xx, yy, Z3, cmap='coolwarm', alpha=.8)
plt.scatter(X[:,0], X[:,1], c = y, cmap = 'coolwarm')
plt.title('15 Neurons')
```

```
Model 1:
4/4 [==============================] - 0s 720us/step - loss: 0.0851 - accuracy:
0.8800
Model 2:
4/4 [==============================] - 0s 680us/step - loss: 0.0351 - accuracy:
```

```
0.9700
Model 3:
4/4 [==============================] - 0s 1ms/step - loss: 0.0011 - accuracy:
1.0000
```

[222]:
```
Text(0.5, 1.0, '15 Neurons')
```



17 Lecture 17 - Facial Expression Classification

[68]:
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
from scipy import ndimage, signal
import pandas as pd
```

Let us load the data first

[69]:
```python
labels = np.loadtxt('true_labels_Facial_train.csv', delimiter=',')
data_train = np.load('data_train_Facial.npy', allow_pickle=True)
data_test = np.load('data_test_Facial.npy', allow_pickle=True)
print(data_train.shape)
print(data_test.shape)
```

```
(20000, 35, 35)
(10000, 35, 35)
```

This is an image data set in form of a numpy array.

It contains images of 35x35 pixels. The images are of faces and the labels correspond to their emotional state:

0: happy

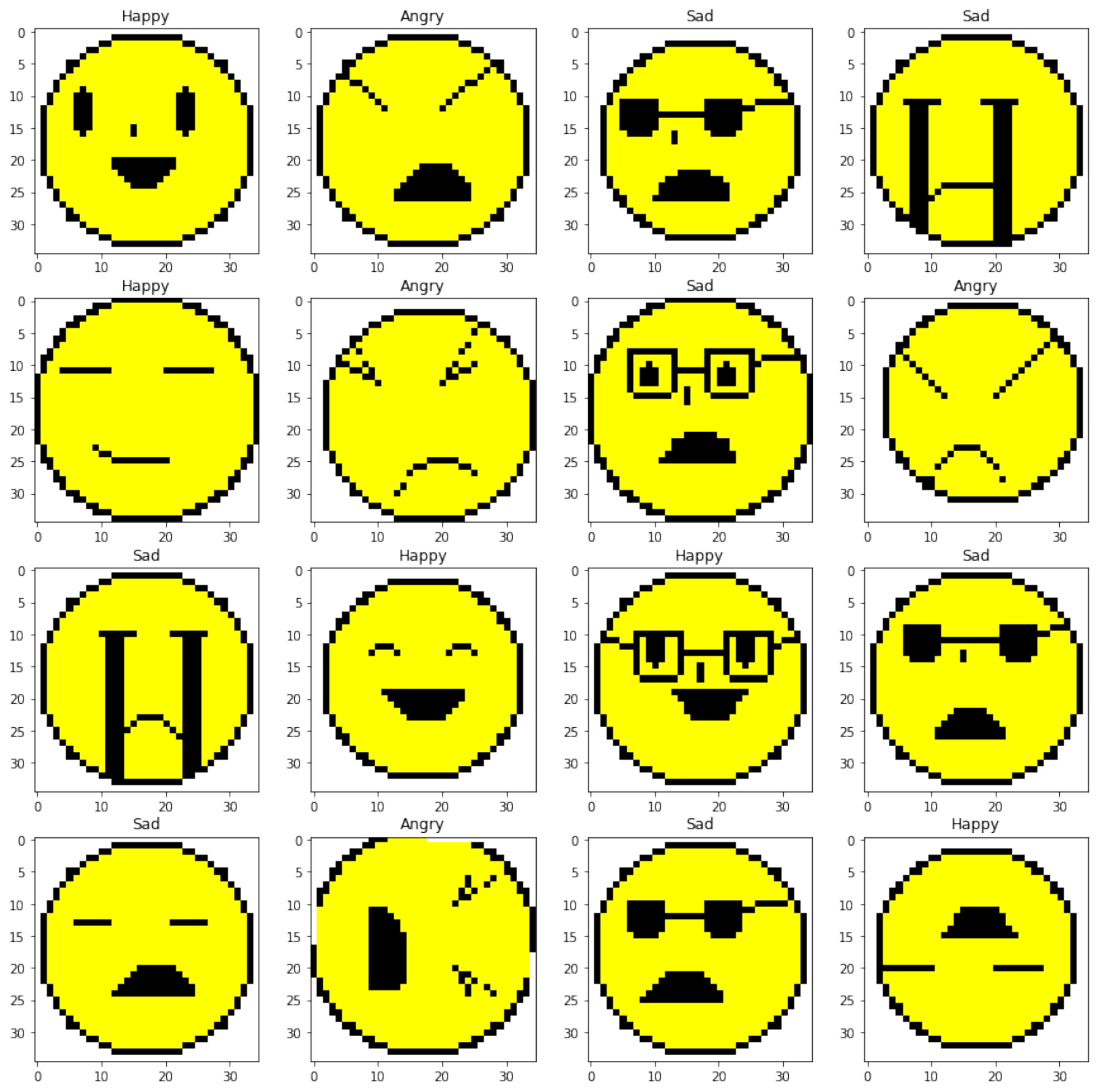
1: sad

2: angry

Lets have a look at the images:

[70]:
```python
cmap = colors.ListedColormap(['white', 'yellow', 'black'])
Emotions = ['Happy', 'Sad', 'Angry']
```

```
plt.figure(figsize = (15,15))
for k in range(16):
    plt.subplot(4,4,k+1)
    plt.imshow(data_train[k,:,:], cmap= cmap)
    plt.title(Emotions[int(labels[k])])
```



Our biggest problem is to deal with the massive input dimension of 35×35.

My solution is to use a very simple algorithm. Other solutions could involve reducing the dimension in a smart way and then applying tools from earlier.

[71]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[98]:
```
# we split the training set into a train and a validation set:
data_train_split = data_train[0:int(data_train.shape[0]/2), :, :]
lab_train_split  = labels[0:int(data_train.shape[0]/2)]
data_validation_split = data_train[int(data_train.shape[0]/2)::, :, :]
lab_validation_split  = labels[int(data_train.shape[0]/2)::]

#we train the nearest neighbor classifier on the training set:
neigh = KNeighborsClassifier(n_neighbors=1)
neigh.fit(np.reshape(data_train_split, [data_train_split.shape[0], data_train_split.shape[1]*data_train.
 ↪shape[2]]), lab_train_split)
```

[98]:
```
KNeighborsClassifier(n_neighbors=1)
```

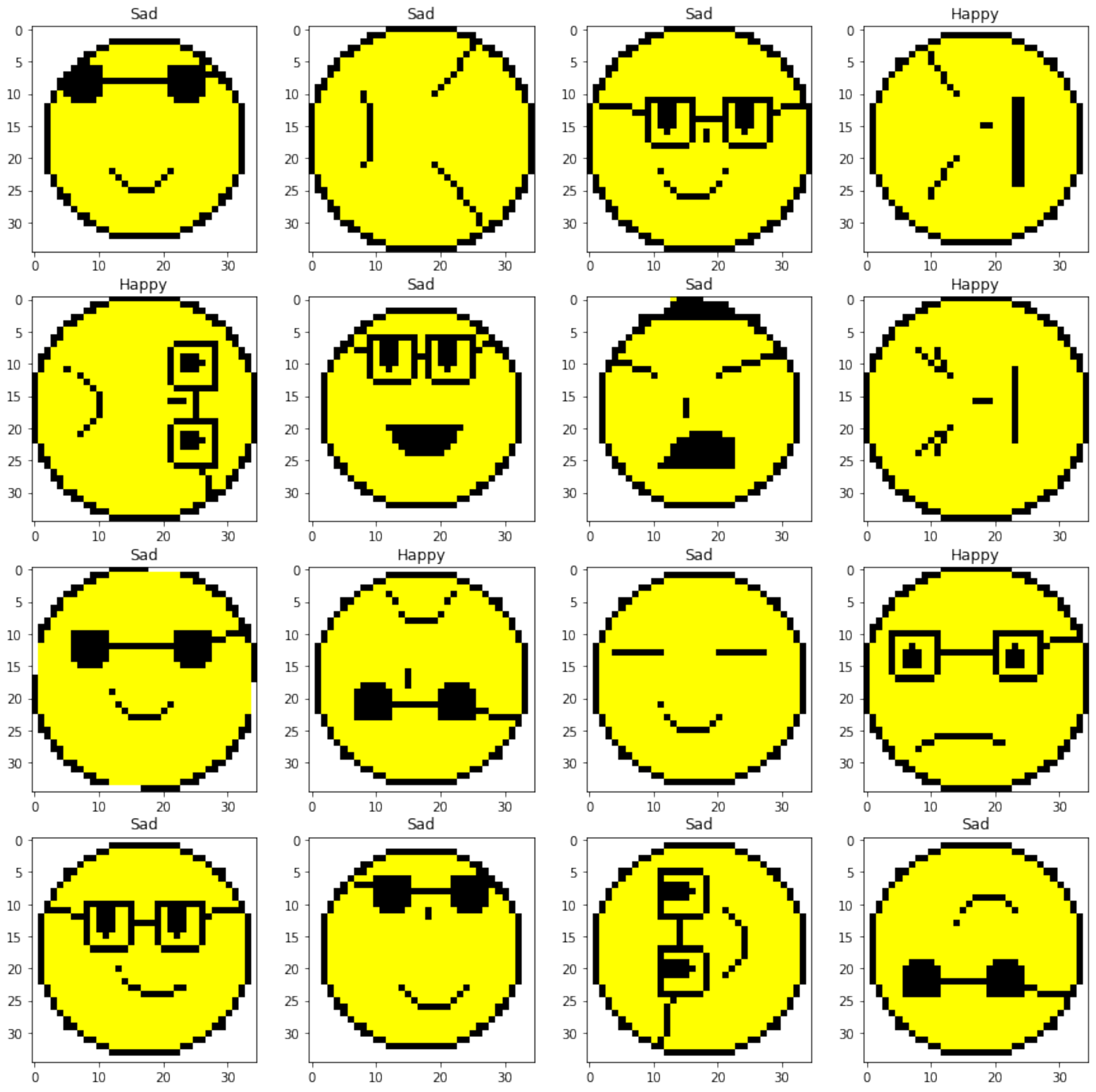
Next we compute the accuracy of our algorithm on the validation set:

[99]:
```
# make prediction:

validation_pred_labels = neigh.predict(np.reshape(data_validation_split, [data_validation_split.
 ↪shape[0], data_validation_split.shape[1]*data_validation_split.shape[2]]))

# validation accuracy:

accuracy = np.sum(lab_validation_split == validation_pred_labels)/lab_validation_split.shape[0]
print('Accuracy: ' + str(accuracy))
```

```
Accuracy: 0.8646
```

Let us have a look at the misclassified data points to see if there is something conspicuous about them.

[100]:
```
# Let's look at some of the missclassified examples:
mistakes = np.where(lab_validation_split != validation_pred_labels)[0]



cmap = colors.ListedColormap(['white', 'yellow', 'black'])
Emotions = ['Happy', 'Sad', 'Angry']

plt.figure(figsize = (15,15))
for k in range(16):
    plt.subplot(4,4,k+1)
    plt.imshow(data_validation_split[mistakes[k],:,:], cmap= cmap)
    plt.title(Emotions[int(validation_labels[mistakes[k]])])
```



I am very happy with the accuracy on the validation set. I also have no simple explanation why the faces above were misclassified and therefore no direct way of improving my algorithm. (One notices a surprisingly high amount of faces with glasses though.) Hence I choose to proceed.

I apply this algorithm to the test set now:

[74]:
```
labels_test = neigh.predict(np.reshape(data_test, [data_test.shape[0], data_test.shape[1]*data_test.
↪shape[2]]))
```

Finally we store the prediction to enter the competition.

[61]:
```
np.savetxt('prediction_facial_recognition_PhilippPetersen.csv', labels_test, delimiter=',')
```

18 Lecture 18 - Boosting

Boosting is a type of ensemble method where multiple classifiers/predictors are combined to yield one more powerful classifier/predictor.

We start with the definition of a weak learning algorithm.

Definition 18.1. *Let $\mathcal{C}$ be a concept class. A* weak PAC learning algorithm *is an algorithm $\mathcal{A}$ taking samples $S \in \mathcal{X}^m$ to functions in $\mathcal{H} \subset \mathcal{X} \times \{-1,1\}$ such that for a $\gamma > 0$ there exists a function $m : (0,1) \to \mathbb{N}$, such that for every $\delta > 0$, all distributions $\mathcal{D}$ on $\mathcal{X}$ and every target concept c, it holds that*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left[\mathcal{R}_S(\mathcal{A}(S)) \leq \frac{1}{2} - \gamma\right] \geq 1-\delta,$$

if $m \geq m(\delta)$.

A weak learning algorithm only needs to be slightly better than the trivial algorithm that predicts Rademacher random labels.

The idea behind boosting is now to cleverly combine the hypotheses returned by weak learning algorithms to build a stronger algorithm.

Probably the most widely-used boosting algorithm is *AdaBoost*:

ADABOOST: *Input:* Base classifier set $\mathcal{H}$, sample $(x_i, y_i)_{i=1}^m$, number of steps T.

```
1. Initialise $\underline{\mathcal{D}}_1$ as the uniform probability distribution on $[m]$.
2. for $t = 1, \dots, T$:
3.     Choose $h_i \in \mathcal{H}$ such that $\epsilon_t := \sum_{i=1}^m \underline{\mathcal{D}}_t(i)\mathbb{1}_{h_t(x_i)\neq y_i}$ is small.
4.     set $\alpha_t := \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)/2$.
5.     for $i = 1, \dots, m$:
6.         set $\underline{\mathcal{D}}_{t+1}(i) := \frac{\underline{\mathcal{D}}_t(i)\exp(-\alpha_t y_i h_t(x_i))}{\sum_{j=1}^m \underline{\mathcal{D}}_t(j)\exp(-\alpha_t y_j h_t(x_j))}$.
7. return $f := \mathrm{sign} \circ \sum_{t=1}^T \alpha_t h_t$.
```

Theorem 18.1. *Let $S = (x_i, y_i)_{i=1}^m$ be a sample, let $\mathcal{H}$ be a set of base classifiers and assume that in the iteration of AdaBoost, $0 < \epsilon_t < 1/2 - \gamma$ for a fixed $\gamma > 0$. Then, for $f = ADABOOST(\mathcal{H}, S, T)$*

$$\widehat{\mathcal{R}}_S(f) = \frac{1}{m}\sum_{i=1}^m \mathbb{1}_{f(x_i)\neq y_i} \leq e^{-2\gamma^2 T}.$$

Proof. Let us denote for $t \in [T]$

$$f_t = \sum_{p\leq t} \alpha_p h_p$$
$$Z_t := \frac{1}{m}\sum_{i=1}^m e^{-y_i f_t(x_i)},$$

and $f_0 = 0$, $Z_0 = 1$. Note that $\mathrm{sign}(f_T) = f = ADABOOST(\mathcal{H}, S, T)$.

Since $\mathbb{1}_{h(x)y \leq 0} \leq e^{-yh(x)}$, we have that

$$\begin{aligned}
\widehat{\mathcal{R}}_S(f) &= \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}_{f(x_i)\neq y_i} \\
&= \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}_{f_T(x_i)y_i \leq 0} \\
&\leq Z_T \\
&= \frac{Z_T}{Z_0} \\
&= \frac{Z_T}{Z_{T-1}} \cdots \frac{Z_1}{Z_0}.
\end{aligned}$$

Therefore, the result follows if we can show that for all $t \in [T-1]$

$$\frac{Z_{t+1}}{Z_t} \leq e^{-2\gamma^2}. \tag{77}$$

Assume that for a fixed $t \in [T-1]$

$$\mathcal{D}_t(i) = \frac{e^{-y_i f_{t-1}(x_i)}}{\sum_{j=1}^m e^{-y_i f_{t-1}(x_j)}}. \tag{78}$$

Then we conclude that

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)\exp(-\alpha_t y_i h_t(x_i))}{\sum_{j=1}^m \mathcal{D}_t(j)\exp(-\alpha_t y_j h_t(x_j))} = \frac{e^{-y_i f_t(x_i)}}{\sum_{j=1}^m e^{-y_i f_t(x_j)}}.$$

Since (78) holds for $t = 1$, we conclude by induction that (78) holds for all $t \in [T]$.

Now we have that

$$\begin{aligned}
\frac{Z_{t+1}}{Z_t} &= \frac{\sum_{i=1}^m e^{-y_i f_{t+1}(x_i)}}{\sum_{i=1}^m e^{-y_i f_t(x_i)}} \\
&= \frac{\sum_{i=1}^m e^{-y_i f_t(x_i)} e^{-y_i \alpha_{t+1} h_{t+1}(x_i)}}{\sum_{i=1}^m e^{-y_i f_t(x_i)}} \\
&= \sum_{i=1}^m \mathcal{D}_{t+1}(i) e^{-y_i \alpha_{t+1} h_{t+1}(x_i)} \\
&= e^{-\alpha_{t+1}} \sum_{i:\, y_i = h_{t+1}(x_i)} + e^{\alpha_{t+1}} \sum_{i:\, y_i \neq h_{t+1}(x_i)} \\
&= e^{-\alpha_{t+1}}(1-\epsilon_{t+1}) + e^{\alpha_{t+1}}\epsilon_{t+1} \\
&= \frac{1}{\sqrt{1/\epsilon_{t+1}-1}}(1-\epsilon_{t+1}) + \sqrt{1/\epsilon_{t+1}-1}\,\epsilon_{t+1} \\
&= \sqrt{\frac{\epsilon_{t+1}}{1-\epsilon_{t+1}}}(1-\epsilon_{t+1}) + \sqrt{\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}}\epsilon_{t+1} \\
&= \sqrt{\epsilon_{t+1}(1-\epsilon_{t+1})} + \sqrt{(1-\epsilon_{t+1})(\epsilon_{t+1})} = 2\sqrt{\epsilon_{t+1}(1-\epsilon_{t+1})}.
\end{aligned}$$

We had assumed that $\epsilon_{t+1} < 1/2 - \gamma$ and hence

$$2\sqrt{\epsilon_{t+1}(1-\epsilon_{t+1})} \leq 2\sqrt{(\gamma-1/2)((\gamma+1/2))} = 2\sqrt{1/4-\gamma^2} = \sqrt{1-4\gamma^2}.$$

Using $1 - x \leq e^{-x}$ yields that

$$\frac{Z_{t+1}}{Z_t} \leq e^{-2\gamma^2}.$$

This completes the proof. $\square$

We saw that Adaboost can very quickly reduce the empirical error, if weak learners exist and can be found quickly. A standard choice for the set of base classifiers is that of so-called *decision stumps* (this name comes from the fact that these are decision trees with minimal depth.), which are linear classifiers acting on a single axis of the data, i.e., for $\mathcal{X} = \mathbb{R}^N$

$$\mathcal{H} := \{x \mapsto b \cdot \operatorname{sign}(x_i - \theta) \colon \theta \in \mathbb{R}, b \in \{\pm 1\}, i \in [N]\}.$$

See Figure 20 for a visualisation of boosting with decision stumps.

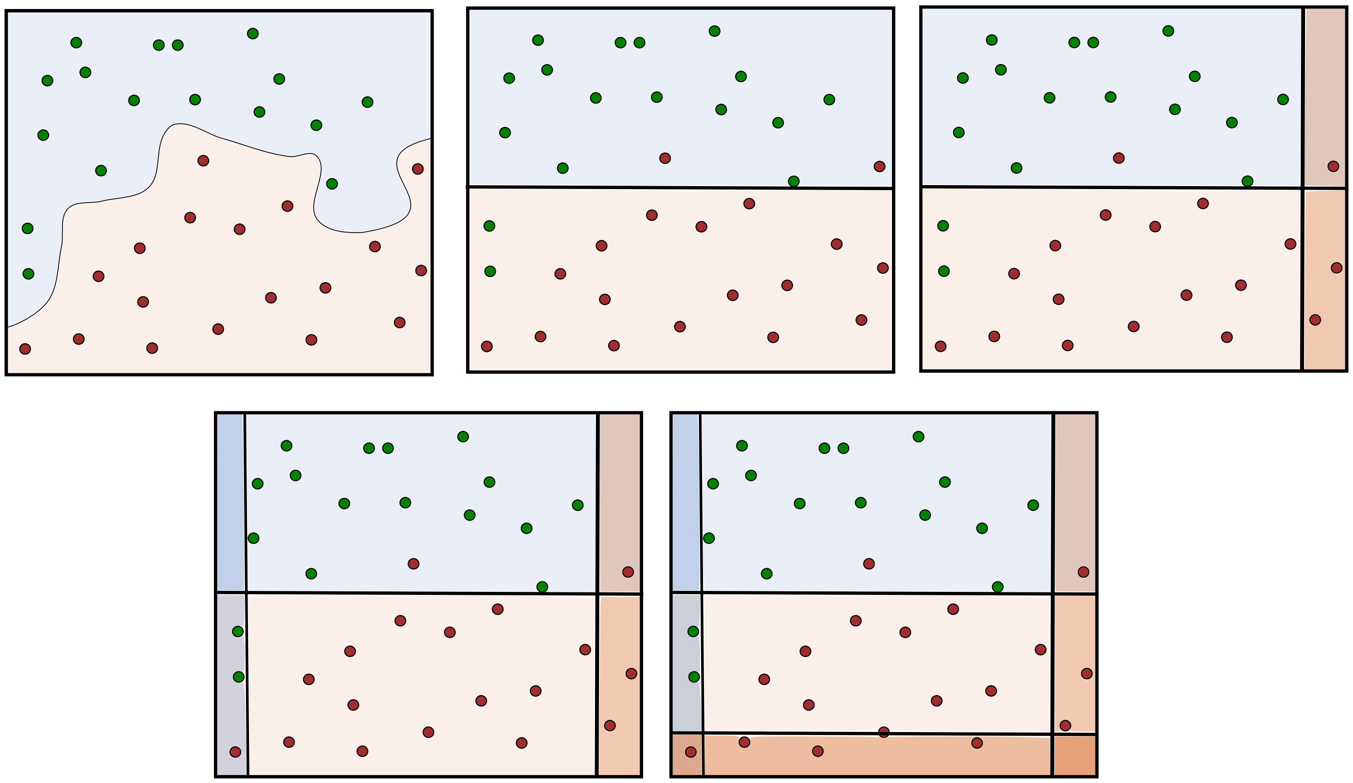


Figure 20: Visualisation of classification with boosting and decision stumps. The top left shows the samples and the underlying disctibution. The next four panels show successively built sums of decision stumps.

Note that the set of decision stumps is quite small. In fact, there exist simple distributions so that there does not exist a PAC learning algorithm with hypothesis set $\mathcal{H}$.

We can ask ourselves how the base class affects the generalisation capabilities of Adaboost. For this, we observe that the output of Adaboost is an element of the following set:

$$L(\mathcal{H}, T) = \left\{ x \mapsto \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x) \right) : \alpha_t \in \mathbb{R}, h_t \in \mathcal{H} \right\}.$$

We can compute the VC dimension of $L(\mathcal{H}, T)$.

Proposition 18.1. *Let $\mathcal{H}$ be a base class and let $T \in \mathbb{N}$, $T \geq 3$. Then*

$$\mathrm{VCdim}(L(\mathcal{H}, T)) \leq 2(d+3)(T+1)\log_2((d+3)(T+1)), \tag{79}$$

where $d := \mathrm{VCdim}(\mathcal{H})$.

Proof. Let $C = (x_1, \dots, x_m)$ be a set of points shattered by $L(\mathcal{H}, T)$.

Every function $f \in L(\mathcal{H}, T)$ is built by the concatenation of $h_1, \dots, h_T$ with a linear classifier. By Theorem 4.3 we have that the set $|\{h(C) \colon h \in \mathcal{H}\}| \leq (em/d)^d = m^d(e/d)^d \leq em^d$, where $d = \mathrm{VCdim}(\mathcal{H})$.

Therefore,

$$|\{(h_1(C), \dots h_T(C)) \colon h_1, \dots, h_T \in \mathcal{H}\}| \leq e^T m^{dT}.$$

By Example 4.3 and Theorem 4.3, we have that for each element in $c = (c_1, \dots, c_T) \in \{(h_1(C), \dots h_T(C)) \colon h_1, \dots, h_T \in \mathcal{H}\}$, the set

$$|\{\mathrm{sign}(\langle a, c\rangle + b) \colon a \in \mathbb{R}^T, b \in \mathbb{R}\}| \leq (em/(T+1))^{T+1} = (e/(T+1))^{T+1} m^{T+1} \leq em^{T+1}.$$

Therefore, we conclude that

$$|\,\{f(C) \colon f \in L(\mathcal{H}, T)\}\,| \leq (e^{T+1} m^{dT + (T+1)}) = e^{T+1} m^{(d+1)T+1} \leq 2^{2(T+1)} m^{(d+1)(T+1)}.$$

Since C was shattered by $L(\mathcal{H}, T)$, we conclude that

$$2^m \leq 2^{2(T+1)} m^{(d+1)(T+1)}$$

and hence

$$m \leq 2(T+1) + (d+1)(T+1)\log_2(m) \leq (d+3)(T+1)\log_2(m). \tag{80}$$

Since for $x > 1$ we have that $\log_2(x) \leq \sqrt{x}$, it follows from (80)

$$\sqrt{m} \leq (d+3)(T+1)$$

and thus

$$\log_2(m) \leq 2\log((d+3)(T+1)). \tag{81}$$

Applying (81) to (80) yields

$$m \leq 2(d+3)(T+1)\log_2((d+3)(T+1)).$$

□

19 Lecture 19 - Clustering

Clustering is the act of associating elements of a data set $(x_i)_{i=1}^m$ into a number of sets that may or may not be determined beforehand. In low dimensions, humans have a very good intuition on how to cluster data points. For example in Figure 21, most people would have a pretty strong opinion on how to cluster the points into 2 or three sets. However, defining a mathematical rule is typically harder. To perform clustering numerically, one needs to specify an objective to minimise or a procedure to follow. We will discuss some examples of such algorithms in this chapter.

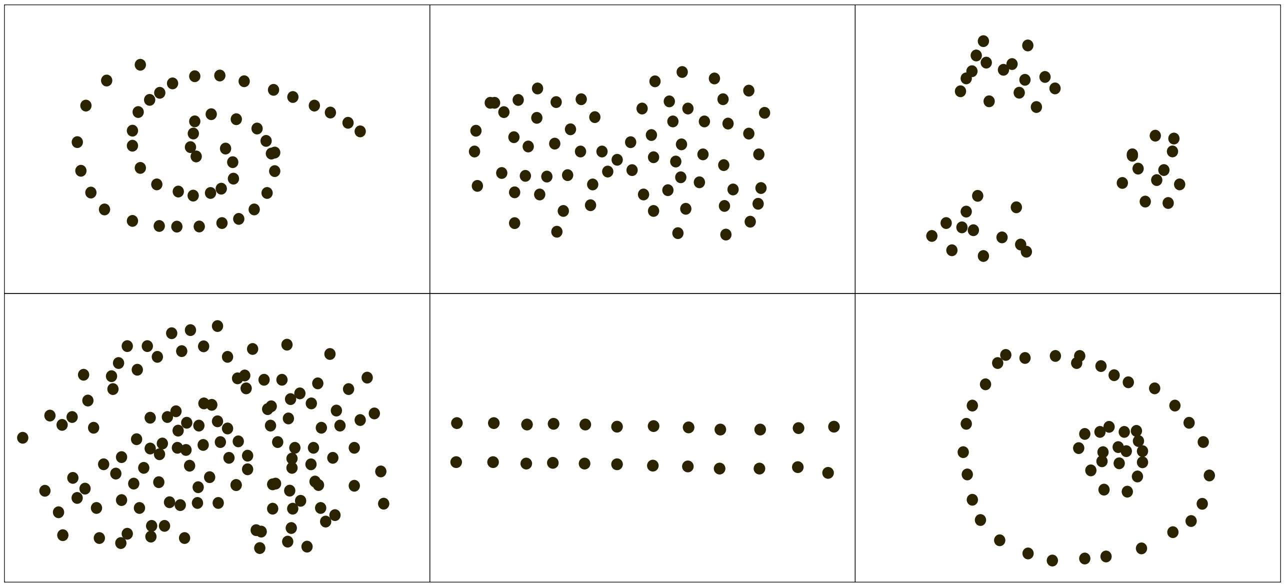


Figure 21: Six clustering problems

Let us first describe the task of clustering in more mathematical terms. Clustering is a procedure that maps an input to an output:

- *Input:* A set $X = (x_i)_{i=1}^m$ and a distance function $d\colon X \times X \to \mathbb{R}^+$ which is symmetric and satisfies $d(x,x) = 0$. Alternatively, a similarity measure $s\colon X \times X \to [0,1]$ can be given with s symmetric and $s(x,x) = 1$.
- *Output:* A sequence of disjoint subsets of X denoted by $(C_i)_{i=1}^k$ such that $\bigcup_{j=1}^k C_k = X$.

How this segmentation of X into the $(C_i)_{i=1}^k$ is performed depends on d or s and is different from algorithm to algorithm.

19.1 Linkage-based clustering

Linkage-based clustering is very simple, but often-times surprisingly effective. It works by the following procedure which is visualised in Figure 22:

1. Start with $(C_i^0)_{i=1}^m \subset X$, disjoint, such that $x_i \in C_i$.
2. Construct, for $\ell < m$ the sets $(C_i^\ell)_{i=1}^{m-\ell}$ by the following procedure: Find i_1, i_2 such that $C_{i_1}^{\ell-1}$ and $C_{i_2}^{\ell-1}$ are the *most similar clusters* (we will discuss what this means below). Then
$$(C_i^\ell)_{i=1}^{m-\ell} = \{C_i^\ell : i \neq i_1, i_2\} \cup \{C_{i_1}^{\ell-1} \cup C_{i_2}^{\ell-1}\}. \tag{82}$$
3. The output of the clustering algorithm is $(C_i^\ell)_{i=1}^{m-\ell}$ for a given ℓ.

The notion of "*most similar clusters*", that was used above can mean many things, depending on the application in mind. A couple of examples are listed below:

- *Single Linkage clustering*: Here we define
$$d(C_i, C_j) := \min\{d(x_\ell, x_{\ell'})\colon x_\ell \in C_i, x_{\ell'} \in C_j, \ell \in [m]\}.$$
- *Average Linkage clustering*: Here we define
$$d(C_i, C_j) := \mathrm{mean}\{d(x_\ell, x_{\ell'})\colon x_\ell \in C_i, x_{\ell'} \in C_j, \ell \in [m]\}.$$

- *Max Linkage clustering*: Here we define

$$d(C_i, C_j) := \max\{d(x_\ell, x_{\ell'}) \colon x_\ell \in C_i, x_{\ell'} \in C_j, \ell \in [m]\}.$$

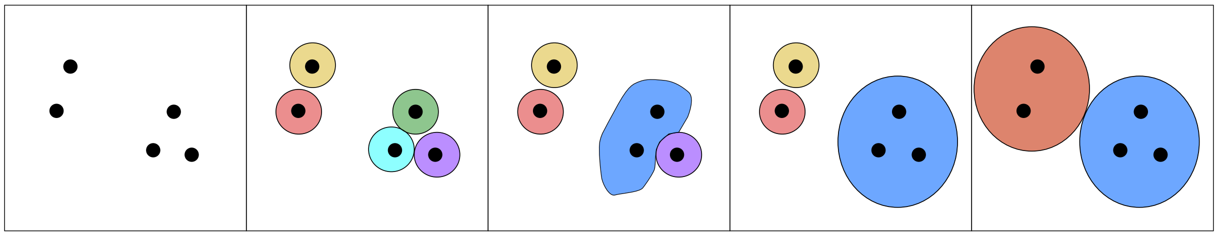


Figure 22: Example of linkage-based clustering.

19.2 k-means clustering

One way to perform clustering is to define a *cost function* which the partition $(C_i)_{i=1}^k$ should minimise. For k-means clustering, we wish to find k clusters such that every point is as close as possible to the center of its associated cluster.

To make this formal, we first define the *centroid* of a cluster C_i as

$$\mu(C_i) := \operatorname{argmin}_{\mu \in \mathrm{co}X} \sum_{x_j \in C_i} d(x_j, \mu)^2,$$

where $(x_j)_{j=1}^m$ is the data set.

Then, k-means clustering consists in finding $(C_i)_{i=1}^k$ minimising

$$G_{k-means}((x_j)_{j=1}^m, d)(C_1, \dots, C_k) := \sum_{i=1}^k \sum_{x_j \in C_i} d(x_j, \mu(C_i))^2. \tag{83}$$

Finding a solution of this problem is NP-hard in general. However, there is a widely used algorithm, that typically performs well. This is Lloyd's algorithm and consists of the following steps:

LLOYD'S ALGORITHM: *Input:* $(x_j)_{j=1}^m$, number of clusters $k \in [m]$.

```
1. Randomly choose initial centroids $\mu_1^1, \dots, \mu_k^1$.
2. while not converged:
3.     $\forall i \in [k]$ set $C_i^{\ell+1} := \{x \in X : i = \operatorname{argmin}_{j \in [k]} \|x - \mu_j^\ell\|\}$,
4.     $\forall i \in [k]$ set $\mu_i^{\ell+1} := \frac{1}{|C_i^{\ell+1}|} \sum_{x \in C_i^{\ell+1}} x$.
5. return the clustering from the last iteration $(C_i^L)_{i=1}^k$.
```

Lemma 19.1. *Each iteration of the k-means algorithm does not increase the k-means objective function $G_{k-means}$ of* (83).

Proof. It holds that for $\bar{\mu}(C) = \frac{1}{|C|}\sum_{x_j \in C} x_j$ that for arbitrary $\lambda \in \mathrm{co}(X)$

$$\begin{aligned}\sum_{x_j \in C} \|x_j - \lambda\|^2 &= \sum_{x_j \in C} \|x_j - \bar{\mu}(C) - (\lambda - \bar{\mu}(C))\|^2 \\ &= \sum_{x_j \in C} \|x_j - \bar{\mu}(C)\|^2 - 2\langle x_j - \bar{\mu}(C), \lambda - \bar{\mu}(C)\rangle + \|\lambda - \bar{\mu}(C))\|^2.\end{aligned}$$

Next, we observe that

$$\begin{aligned}&\sum_{x_j \in C} \|x_j - \bar{\mu}(C)\|^2 - 2\langle x_j - \bar{\mu}(C), \lambda - \bar{\mu}(C)\rangle + |\lambda - \bar{\mu}(C))|^2 \\ &\quad = \left(\sum_{x_j \in C} \|x_j - \bar{\mu}(C)\|^2\right) - 2\left\langle \sum_{x_j \in C} (x_j - \bar{\mu}(C)), \lambda - \bar{\mu}(C)\right\rangle + \|\lambda - \bar{\mu}(C))\|^2 \\ &\quad = \left(\sum_{x_j \in C} \|x_j - \bar{\mu}(C)\|^2\right) + \|\lambda - \bar{\mu}(C))\|^2 \geq \sum_{x_j \in C} \|x_j - \bar{\mu}(C)\|^2.\end{aligned}$$

Hence $\bar{\mu}(C) = \mu(C)$. Let for $\ell \in \mathbb{N}$ and $i \in [k]$, C_i^ℓ be as in Lloyd's algorithm. We have that

$$\begin{aligned}G_{k-means}((x_j)_{j=1}^m, d)(C_1^{\ell+1}, \dots, C_k^{\ell+1}) &= \sum_{i=1}^k \sum_{x_j \in C_i^{\ell+1}} \|x_j - \mu(C_i^{\ell+1})\|^2 \\ &\leq \sum_{i=1}^k \sum_{x_j \in C_i^{\ell+1}} \|x_j - \mu_i^\ell\|^2 \\ &\leq \sum_{i=1}^k \sum_{x_j \in C_i^{\ell}} \|x_j - \mu_i^\ell\|^2 \\ &= \sum_{i=1}^k \sum_{x_j \in C_i^{\ell}} \|x_j - \mu(C_i^\ell)\|^2 = G_{k-means}((x_j)_{j=1}^m, d)(C_1^\ell, \dots, C_k^\ell),\end{aligned}$$

where the first inequality follows by the definition of $\mu(C_i^{\ell+1})$ as a minimum, the second inequality follows from the definition of $\mu(C_i^\ell)$, and penultimate equality follows by the considerations at the beginning of the proof. $\square$

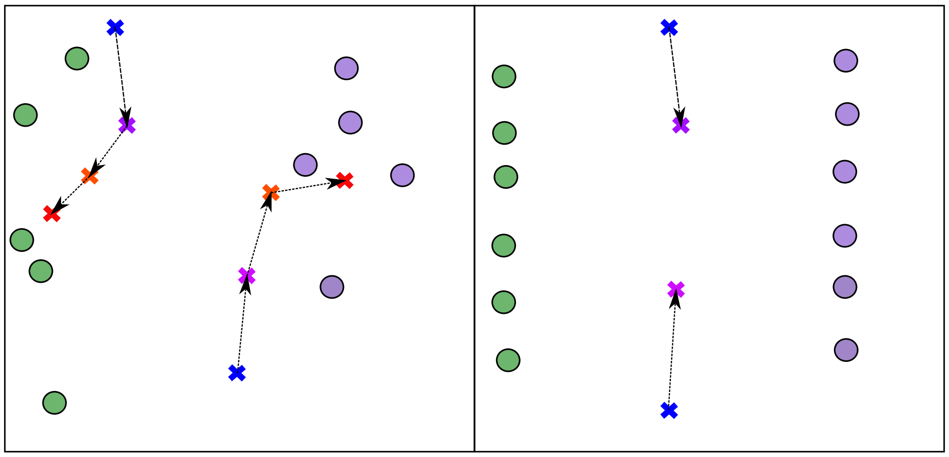


Figure 23: Two examples of the evolution of means μ_1^ℓ, μ_2^ℓ in Lloyd's algorithm. On the left-hand side, we observe successful convergence. On the right-hand side, there is no convergence.

Remark 19.1. *Lloyd's algorithm is simple and very often effective. However, it comes with a couple of issues. First of all, convergence is not guaranteed or could take a very long time. An example of a bad initialisation prohibiting convergence is shown in Figure 23. Moreover, k-NN generally suffers from the issue that all clusters must necessarily be convex in the sense that if $x \in \operatorname{co}(C_i) \cap X$, then $x \in C_i$. In some of the examples of Figure 21, this can be a serious issue. See also Figure 24 for an illustration.*

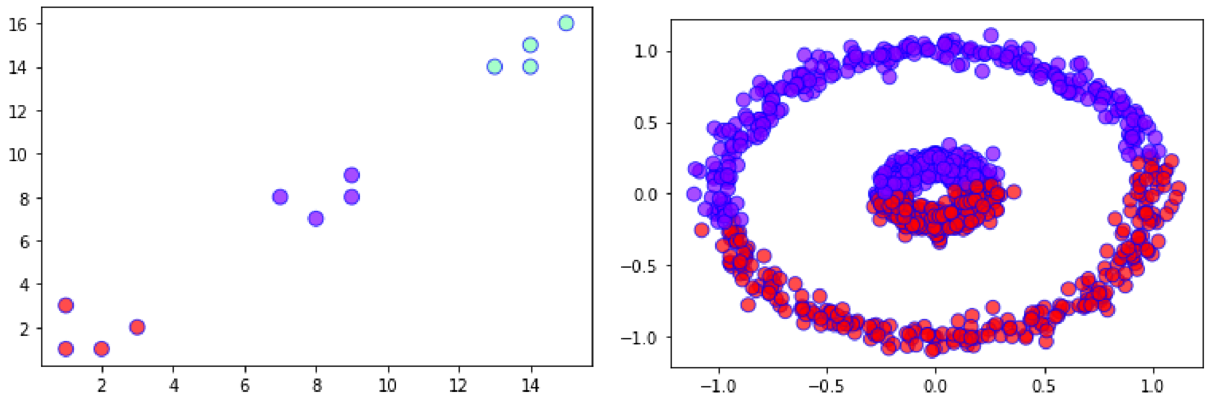


Figure 24: K-means clustering for two data sets. On the left hand side the clustering (with $k = 3$) was successful. The problem on the right hand side cannot be clustered correctly (with $k = 2$), because this would require non-convex clusters.

19.3 Spectral clustering

It is often convenient to cast a clustering problem in the framework of graph theory. In this setting, the relationship between data points is described by weights associated to edges between them. Clustering is then the task of splitting the graph in multiple subgraphs according to some rules. Let us first make a formal definition of a graph.

Definition 19.1. *An undirected* graph *$G = (V, E)$ is a tuple of a* set of nodes *$V = (v_1, \dots, v_n)$ and edges $E \subset \{(i,j), i,j \in [n]\}$, such that for all $i,j \in [n]$, $(i,j) \in E$ if $(j,i) \in E$. We say that v_i, v_j are connected if $(i,j) \in E$.*
A graph is often represented through its adjacency matrix *$A \in \mathbb{R}^{n\times n}$, defined as*

$$A_{i,j} := \begin{cases} 1 & \text{if } (i,j) \in E, \\ 0 & \text{else }. \end{cases}$$

If one wants to make a more nuanced description of the relationship between data points, then a *weighted graph* is more appropriate.

Definition 19.2. *A* weighted graph *is a triple (V, E, W), where (V, E) is an undirected graph and $W \in \mathbb{R}^{n\times n}$ is a symmetric matrix with positive entries and $n = |V|$.*

Example 19.1. *In Figure 25, we show two graphs with the following adjacency matrices:*

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \quad \textit{and} \quad \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{84}$$

Next, we would like to define a measure that allows us to formulate an objective for clustering. A natural measure is the so-called *cut*.

Definition 19.3. *Let (V, E, W) be a weighted graph. Let $S \subset V$ be a vertex partition. We define the* cut of S *as*

$$\text{cut}(S) := \sum_{v_i \in S} \sum_{v_j \in S^c} w_{i,j}.$$

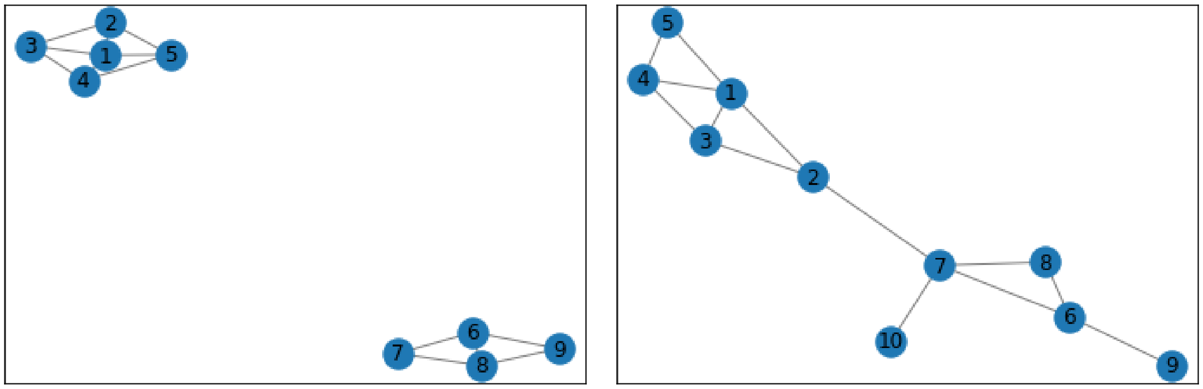


Figure 25: Two visualisations of the simple graphs of Example 19.1

A small cut, means that not many large weights needed to be removed in order to make the partition. Intuitively, this seems to be a reasonable condition for a clustering algorithm. It turns out that the cut is closely related to the so-called *graph Laplacian* of a graph.

Definition 19.4. *Let $G = (V, E, W)$ be a weighted graph. The* degree matrix *of G is defined as*

$$D_{i,i} = \deg(i) := \sum_{(i,j) \in E} w_{i,j}.$$

The graph Laplacian *of G is given by*

$$L_G = D - W.$$

Remark 19.2. *The graph Laplacian L_G satisfies the following formula:*

$$L_G = \sum_{i<j} w_{i,j} (e_i - e_j)(e_i - e_j)^T, \tag{85}$$

where e_i, e_j are the canonical unit (column) vectors.

Let S be a vertex partition and $y \in \{\pm 1\}^n$ be such that $y_i = 1$ if and only if $v_i \in S$. Then it holds that

$$\frac{1}{4}\sum_{i<j} w_{i,j}(y_i - y_j)^2 = \frac{1}{4}\left(4 \sum_{i\in S, j\in S^c, i<j} w_{i,j} + 4 \sum_{i\in S^c, j\in S, i<j} w_{i,j}\right)$$
$$= \left(\sum_{i\in S, j\in S^c, i<j} w_{i,j} + \sum_{i\in S, j\in S^c, i>j} w_{i,j}\right) = \mathrm{cut}(S).$$

Using the formula above, we are now able to express the cut in terms of the graph Laplacian.

Proposition 19.1. *Let $G = (V, E, W)$ be a weighted graph and let L_G be the associated graph Laplacian. It holds for all $x \in \mathbb{R}^n$, where $n = |V|$ that*

$$x^T L_G x = \sum_{i<j} w_{i,j}(x_i - x_j)^2. \tag{86}$$

In particular,

$$\mathrm{cut}(S) = \frac{1}{4} y^T L_G y, \tag{87}$$

for $y \in \{\pm 1\}$ with $y_i = 1$ if and only if $v_i \in S$.

Proof. By associativity we have that

$$\sum_{i<j} w_{i,j}(x_i - x_j)^2 = \sum_{i<j} w_{i,j}((e_i - e_j)^T x)^2$$
$$= \sum_{i<j} w_{i,j}(x^T(e_i - e_j))((e_i - e_j)^T x)$$
$$= \sum_{i<j} w_{i,j} x^T (e_i - e_j)(e_i - e_j)^T x.$$

Using now the linearity as well as (85) yields that

$$\sum_{i<j} w_{i,j}(x_i - x_j)^2 = x^T L_G x.$$

□

Remark 19.3. *Proposition 19.1 shows that we can compute the cut by computing a quadratic form involving the graph Laplacian. Minimising expressions of the form $x^T A x$ for positive semidefinite matrices reduces to an eigenvalue problem and can be considered simple. We have two issues though: First, we do not want to minimise such an expression over general $x \in \mathbb{R}^n$, but only over x that take values in $\{\pm 1\}$. Second, a minimiser of the cut is actually very easy to compute. In fact $S = \emptyset$ or $S = V$ always yield* $\mathrm{cut}(S) = 0$*, the minimal value. This is, however, not the minimiser we were looking for.*

To address the issues raised in Remark 19.3, we first introduce a different notion of a cut that promotes balanced partitions.

Definition 19.5. *Let $G = (V, E, W)$ be a weighted graph and $S \subset V$ be a vertex partition. We define the* weighted cut *of G as*

$$\mathrm{Ncut}(S) := \frac{\mathrm{cut}(S)}{\mathrm{vol}(S)} + \frac{\mathrm{cut}(S^c)}{\mathrm{vol}(S^c)} = \mathrm{cut}(S)\left(\frac{1}{\mathrm{vol}(S)} + \frac{1}{\mathrm{vol}(S^c)}\right),$$

where $\mathrm{vol}(S) = \sum_{i\in S} \deg(i)$*. Here one needs to decide on a convention for $S = \emptyset$ or $S = V$.*

The following proposition holds:

Proposition 19.2. *Let $G = (V, E, W)$ be a weighted graph and $S \subset V$ be a vertex partition. It holds that*

$$\mathrm{Ncut}(S) = y^T L_G y$$

where

$$y_i = \begin{cases} \left(\frac{\mathrm{vol}(S^c)}{\mathrm{vol}(S)\mathrm{vol}(V)}\right)^{1/2} & \text{if } i \in S, \\ -\left(\frac{\mathrm{vol}(S)}{\mathrm{vol}(S^c)\mathrm{vol}(V)}\right)^{1/2} & \text{if } i \in S^c. \end{cases}$$

Proof. It holds by (86) that

$$\begin{aligned} y^T L_G y &= \frac{1}{2} \sum_{i,j \in V} w_{i,j}(y_i - y_j)^2 \\ &= \sum_{i \in S} \sum_{j \in S^c} w_{i,j}(y_i - y_j)^2 \\ &= \sum_{i \in S} \sum_{j \in S^c} w_{i,j} \left(\left(\frac{\mathrm{vol}(S^c)}{\mathrm{vol}(S)\mathrm{vol}(V)} \right)^{1/2} + \left(\frac{\mathrm{vol}(S)}{\mathrm{vol}(S^c)\mathrm{vol}(V)} \right)^{1/2} \right)^2 \\ &= \sum_{i \in S} \sum_{j \in S^c} w_{i,j} \left(\frac{\mathrm{vol}(S^c)}{\mathrm{vol}(S)\mathrm{vol}(V)} + 2 \left(\frac{\mathrm{vol}(S^c)}{\mathrm{vol}(S)\mathrm{vol}(V)} \frac{\mathrm{vol}(S)}{\mathrm{vol}(S^c)\mathrm{vol}(V)} \right)^{1/2} + \frac{\mathrm{vol}(S)}{\mathrm{vol}(S^c)\mathrm{vol}(V)} \right) \\ &= \sum_{i \in S} \sum_{j \in S^c} w_{i,j} \left(\frac{\mathrm{vol}(S^c)}{\mathrm{vol}(S)\mathrm{vol}(V)} + \frac{2}{\mathrm{vol}(V)} + \frac{\mathrm{vol}(S)}{\mathrm{vol}(S^c)\mathrm{vol}(V)} \right) \\ &= \sum_{i \in S} \sum_{j \in S^c} \frac{w_{i,j}}{\mathrm{vol}(V)} \left(\frac{\mathrm{vol}(S^c)}{\mathrm{vol}(S)} + 2 + \frac{\mathrm{vol}(S)}{\mathrm{vol}(S^c)} \right). \end{aligned}$$

Using that $\frac{\mathrm{vol}(S)}{\mathrm{vol}(S)} + \frac{\mathrm{vol}(S^c)}{\mathrm{vol}(S^c)} = 2$, we obtain that

$$\begin{aligned} y^T L_G y &= \sum_{i \in S} \sum_{j \in S^c} \frac{w_{i,j}}{\mathrm{vol}(V)} \left(\frac{\mathrm{vol}(S^c) + \mathrm{vol}(S)}{\mathrm{vol}(S)} + \frac{\mathrm{vol}(S) + \mathrm{vol}(S^c)}{\mathrm{vol}(S^c)} \right) \\ &= \sum_{i \in S} \sum_{j \in S^c} w_{i,j} \left(\frac{1}{\mathrm{vol}(S)} + \frac{1}{\mathrm{vol}(S^c)} \right) = \mathrm{Ncut}(S). \end{aligned}$$

□

Using Proposition 19.3, we can now rewrite the minimisation of Ncut as a minimisation problem involving the graph Laplacian.

$$\begin{aligned} &\min y^T L_G y \quad \text{s.t.} \\ &y \in \{a, b\}^n, \text{ for some } a, b \in \mathbb{R}, \\ &y^T D y = 1, \\ &y^T D 1 = 0. \end{aligned} \tag{88}$$

Proposition 19.3. *Let $G = (V, E, W)$ be a weighted graph and $S \subset V$ be a vertex partition. S is a minimum of* Ncut *if and only if y is a minimiser of (88) with $y_i \in \{a, b\}$ for some $a, b \in \mathbb{R}$ and all $i \in [n]$ and $S = \{i \in [n] \colon y_i = a\}$.*

Proof. We only need to show that, for every minimiser y of (88), the entries are of the form prescribed in Proposition 19.3 as well as that every y of the form given by Proposition 19.3 satisfies the constraints of (88). This is left as an exercise for the reader. ☐

Unfortunately, minimising (88) is in not simple at all. In fact, it can be shown to be an NP hard problem. However, we can simplify this problem by relaxing the condition that y can only take two values.

$$\begin{aligned}
&\min y^T L_G y \quad \text{s.t.} && (89)\\
&y \in \mathbb{R}^n, && (90)\\
&y^T D y = 1,\\
&y^T D 1 = 0.
\end{aligned}$$

This optimisation problem is simple. In fact it is equivalent to an eigenvalue problem. Set $z = D^{1/2}y$, then we obtain the problem

$$\begin{aligned}
&\min z^T \mathcal{L}_G z \quad \text{s.t.} && (91)\\
&z \in \mathbb{R}^n,\\
&\|z\|_2 = 1,\\
&(D^{1/2}1)^T z = 0,
\end{aligned}$$

where $\mathcal{L}_G = D^{-1/2} L_G D^{-1/2}$. To solve (91), we first observe that

$$\mathcal{L}_G D^{1/2} 1 = D^{-1/2} L_G 1 = D^{-1/2}(D - W)1 = 0.$$

Hence, $D^{1/2}1$ is an eigenvector associated to the smallest eigenvalue of $\mathcal{L}_G$. We recall the following consequence of the Courant-Fischer Theorem:

Theorem 19.1. *For a matrix $A \in \mathbb{R}^{n\times n}$ it holds that*

$$\lambda_2(A) = \min_{\|x\|=1, x\perp v_1} x^T A x = v_2^T A v_2,$$

where λ_2 is the second smallest eigenvalue of A and v_2 is an associated eigenvector. Moreover, v_1 is the eigenvector associated to the smallest eigenvalue of A.

We conclude that the solution of (91) is the smallest eigenvector associated to the second smallest eigenvalue of $\mathcal{L}_G$.

This is the motivaton for the following algorithm:

SPECTRAL CLUSTERING: *Input:* Graph $G = (V, E, W)$, threshhold $\tau \in \mathbb{R}$.

```
1. construct $\mathcal{L}_G = D^{-1/2}(D - W)D^{-1/2}$.
2. compute $\varphi_2 = D^{-1/2} v_2$.
3. set $S_\tau := \{i \in V : \phi_2(i) \le \tau\}$
4. **return** $S_\tau$.
```

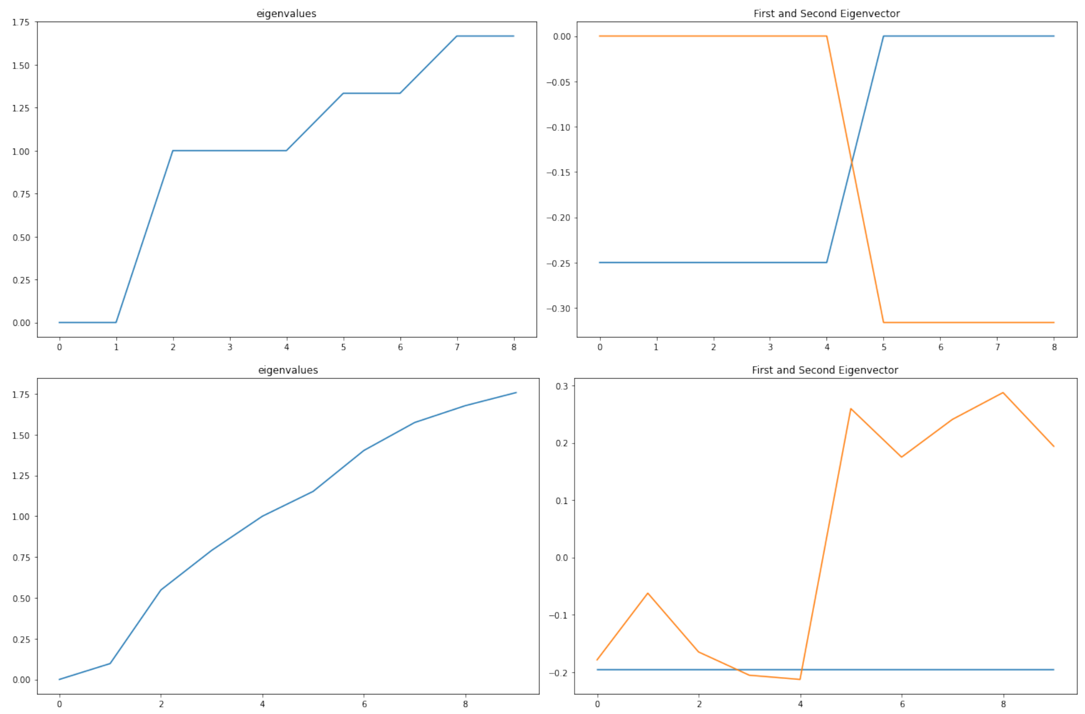


Figure 26: Eigenvalues and $D^{-1/2}v_1$ and $D^{-1/2}v_2$ for v_1, v_2 the eigenvectors associated to the smallest and second smallest eigenvalue of $\mathcal{L}_G$ for the two graphs of Figure 25.

The relationship of spectral clustering to the problem (88) or equivalently the minimisation of the Ncut problem is given by the following result:

Theorem 19.2. *Let $G = (V, E, W)$ be a weighted graph. There exists a threshhold $\tau \in \mathbb{R}$ such that*

$$\lambda_2(\mathcal{L}_G) \lesssim \mathrm{Ncut}(S_\tau) \lesssim \sqrt{\lambda_2(\mathcal{L}_G)},$$

where S_τ is the result of spectral clustering with parameter τ. Here all implicit constants are less than 4.

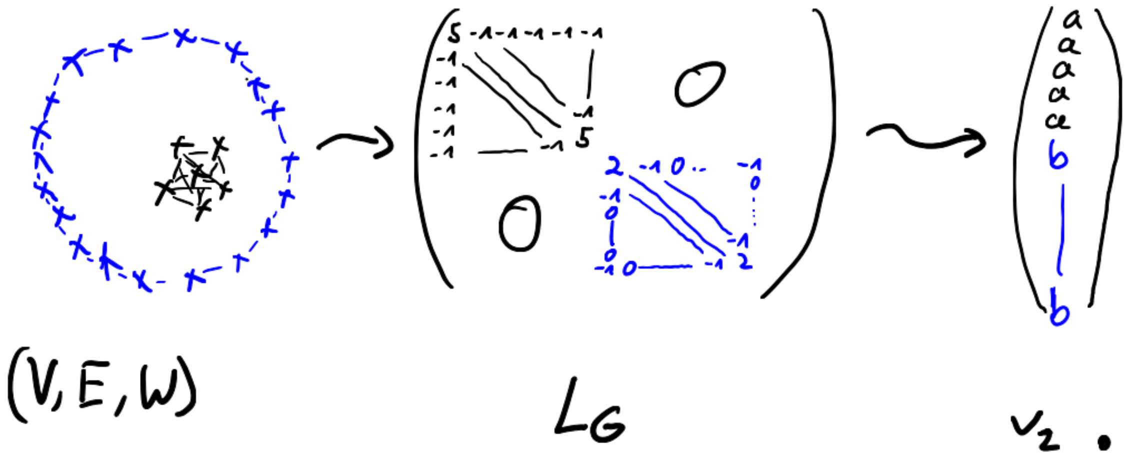


Figure 27: Spectral clustering is successful for the non-convex clustering problem above.

20 Lecture 20 - Dimensionality reduction

We have run into the curse of dimension when we analysed the k-nearest neighbour algorithm. We found that, for some algorithms to work, it is beneficial if the input dimension is small. Sometimes it is possible to reduce the dimension of the data space, without really compromising the data. For example, if the data lies in a low dimensional subspace, it is conceivable that we could restrict our learning problem to this low dimensional subspace and thereby simplify it. A bit more involved is the situation if the data only lies on or close to a low-dimensional non-linear manifold. In both situations, we cannot certainly, still need to find the low dimensional structure before reducing the problem to the simplified setup. Doing this is called *dimensionality reduction*.

20.1 Principle component analysis

We assume that we were given $n \in \mathbb{N}$ data points $(x_i)_{i=1}^n$ which lie in a high dimensional space $\mathbb{R}^d$. For some reason, we believe that we can also represent these data points in a p- dimensional space, where $p < d$. Our idea is to project onto a suitable p- dimensional affine subspace of $\mathbb{R}^d$. One way of doing this is by looking for orthogonal vectors $v_1, \dots, v_p \in \mathbb{R}^d$ a vector $\mu \in \mathbb{R}^d$ as well as coefficients $(\beta_i)_{i=1}^n \in \mathbb{R}^p$ such that

$$x_i \approx \mu + \sum_{k=1}^p (\beta_i)_k v_k = \mu + V\beta_i,$$

where V is the matrix with k-th row equal to v_k^T. We only need to decide what we mean by $\approx$. In the case of principle component analysis (PCA), we choose $\mu, V, (\beta_i)_{i=1}^n$ such that the least squares error is small. Concretely, we are looking for μ, V, β that assume the following minimum:

$$\min_{\substack{\mu, V, \beta_k \\ V^T V = \mathrm{Id}}} \sum_{i=1}^n \|x_i - \mu - V\beta_i\|_2^2. \tag{92}$$

Finding the minimiser of (96) can be done in multiple steps. First we restrict ourselves to minimisers of the form $\sum_{i=1}^n \beta_i = 0$. Indeed, if $\sum_{i=1}^n \beta_i = \lambda \neq 0$ then we can replace β_i by $\tilde{\beta} = \beta_i - \lambda/n$ and observe that

$$\sum_{i=1}^n \|x_i - \mu - V\beta_i\|_2^2 = \sum_{i=1}^n \|x_i - \tilde{\mu} - V\tilde{\beta}_i\|_2^2. \tag{93}$$

where $\tilde{\mu} = \mu - \lambda/nV1$ and 1 denotes the vector with all entries equal to 1.

Under this assumption on the β's, we first seek to find μ. This is done by looking for a stationary point of (96) in μ, i.e., μ^* such that

$$\nabla_\mu \sum_{i=1}^n \|x_i - \mu^* - V\beta_i\|_2^2 = 0. \tag{94}$$

We compute:

$$\nabla_\mu \sum_{i=1}^n \|x_i - \mu - V\beta_i\|_2^2 = -2\sum_{i=1}^n (x_i - \mu - V\beta_i) = 2n\mu - 2\sum_{i=1}^n x_i + V\sum_{i=1}^n \beta_i = 2n\mu - 2\sum_{i=1}^n x_i.$$

Combining the computation above with (94) yields that $\mu^* = \frac{1}{n}\sum_{i=1}^n x_i$ is the sample mean of the data points.

We find the β_i's next. Since V is orthogonal, for a fixed $i \in [n]$, the solution of

$$\min_{\beta_i} \|x_i - \mu - V\beta_i\|_2^2$$

is given by $\beta_i = V^T(x_i - \mu)$. We simply need to show that this choice of β_i satisfies $\sum_{i=1}^n \beta_i = 0$. This of course follows immediately from the linearity of V^T.

In the final step, we would like to find V. By the previous computations the problem reduces to

$$\min_{V^TV=\mathrm{Id}} \sum_{i=1}^n \|x_i - \mu^* - VV^T(x_i - \mu^*)\|_2^2. \tag{95}$$

The binomial formula yields that

$$\begin{aligned}\|x_i - \mu^* - VV^T(x_i - \mu^*)\|_2^2 &= (x_i-\mu^*)^T(x_i-\mu^*) - 2(x_i-\mu^*)^TVV^T(x_i-\mu^*) + (x_i-\mu^*)^TVV^TVV^T(x_i-\mu^*)\\ &= (x_i-\mu^*)^T(x_i-\mu^*) - (x_i-\mu^*)^TVV^T(x_i-\mu^*),\end{aligned}$$

where we used $V^TV = \mathrm{Id}$. The first term above does not depend on V and so we observe that the optimisation problem (95) is equivalent to

$$\max_{V^TV=\mathrm{Id}} \sum_{i=1}^n (x_i-\mu^*)^TVV^T(x_i-\mu^*). \tag{96}$$

Now we use some of the magic of the trace operator. We denote by $\mathrm{Tr}(A) = \sum_{i=1}^k A_{ii}$ the trace operator. Note that for a scalar $\lambda \in \mathbb{R}$ it holds that $\mathrm{Tr}(\lambda) = \lambda$. It is also well known that for matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times m}$ it holds that $\mathrm{Tr}(BA) = \mathrm{Tr}(AB)$.

Also, directly from the definition, we have that $\mathrm{Tr}(A) = \mathrm{Tr}(A^T)$. It holds that

$$\begin{aligned}\max_{V^TV=\mathrm{Id}} \sum_{i=1}^n (x_i-\mu^*)^TVV^T(x_i-\mu^*) &= \max_{V^TV=\mathrm{Id}} \sum_{i=1}^n \mathrm{Tr}\left((x_i-\mu^*)^TVV^T(x_i-\mu^*)\right)\\ &= \max_{V^TV=\mathrm{Id}} \sum_{i=1}^n \mathrm{Tr}\left(V^T(x_i-\mu^*)(x_i-\mu^*)^TV\right)\\ &= \max_{V^TV=\mathrm{Id}} (n-1)\mathrm{Tr}\left(V^T\Sigma_nV\right),\end{aligned} \tag{97}$$

where $\Sigma_n = \frac{1}{n-1}\sum_{i=1}^n (x_i-\mu^*)(x_i-\mu^*)^T$ is the sample variance. It is not hard to see that $\mathrm{Tr}\left(V^T\Sigma_nV\right)$ is maximised by choosing $V = (v_1, \ldots, v_p)$, where $v_1, \ldots, v_p$ are the p eigenvectors associated to the p largest eigenvalues of Σ_n.

We introduced PCA in (96) as the affine subspace that best approximates the data points in a least squares sense. Interestingly, there is a second interpretation of PCA. Indeed, we can equivalently characterise the subspace of PCA as that which maximises the covariance of the data points.

Indeed

$$\begin{aligned}\max_{V^TV=\mathrm{Id}} \sum_{k=1}^n \|V^Tx_i - \frac{1}{n}\sum_{r=1}^n V^Tx_r\|_2^2 &= \max_{V^TV=\mathrm{Id}} \sum_{k=1}^n \|V^T(x_i-\mu^*)\|_2^2\\ &= \max_{V^TV=\mathrm{Id}} (n-1)\mathrm{Tr}\left(V^T\Sigma_nV\right),\end{aligned}$$

where we applied the computation (97) in the last equation.

20.2 Johnson-Lindenstrauss embedding

Assume we have n points in $\mathbb{R}^d$, which we call $(x_i)_{i=1}^n$. Now we want to find a low dimensional representation of the $(x_i)_{i=1}^n$ such that the pairwise distances of the x_i are not distorted by too much. Let us make the phrase "not distorted by too much" a bit more precise.

Definition 20.1. *For $(x_i)_{i=1}^n \subset \mathbb{R}^d$ and $\epsilon \geq 0$ we call a map $f\colon \mathbb{R}^d \to \mathbb{R}^p$ an ϵ-isometry if for all $i, j \in [n]$*

$$(1-\epsilon)\|x_i - x_j\|^2 \leq \|f(x_i) - f(x_j)\|^2 \leq (1+\epsilon)\|x_i - x_j\|^2. \tag{98}$$

Now it is clear, that a 0-isometry exists if $p \geq \min\{d, n\}$, since in this case $(x_i)_{i=1}^n$ lie in a p dimensional space and we can simply project onto this space without distorting the pairwise distances at all.

Nonetheless, the question arises, how small p can be to still allow for an ϵ-isometry.

The following theorem yields a lower bound on p such that a *linear* ϵ-isometry exists.

Theorem 20.1. *Let $n, d, p \in \mathbb{N}$, $d \geq 4$, and $0 < \epsilon < 1/2$ be such that*

$$p \geq \frac{20}{\epsilon^2} \log n.$$

Then, for any set $(x_i)_{i=1}^n \subset \mathbb{R}^d$, there exists a linear ϵ-isometry $f\colon \mathbb{R}^d \to \mathbb{R}^p$.

The proof of this theorem is based on a probabilistic argument. We will need the following concentration inequality.

Lemma 20.1 ([5, Lemma 15.3]). *Let $x \in \mathbb{R}^d$, $p < d$ and assume that A is a $p \times d$ matrix with every entry independently normally distributed. Then it holds that for every $0 < \epsilon < 1/2$*

$$\mathbb{P}\left[(1-\epsilon)\|x\|^2 \leq \left\|\frac{1}{\sqrt{p}}Ax\right\|^2 \leq (1+\epsilon)\|x\|^2\right] \geq 1 - 2e^{-(\epsilon^2-\epsilon^3)p/4}.$$

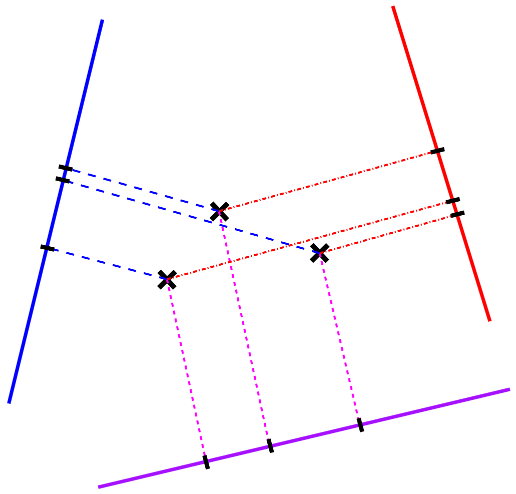


Figure 28: Projection of 3 points onto three subspaces. The projection onto the purple subspace distorts the pairwise distances the least.

Proof of Theorem 20.1. Let p be as in the statement of the theorem and choose $f = \frac{1}{\sqrt{p}}A$, where A is a $p \times d$ matrix with all entries i.i.d normal distributed. Let x_i, x_j be two points in $(x_i)_{i=1}^n$ then it holds that with probability at least $1 - 2e^{-(\epsilon^2-\epsilon^3)p/4}$

$$(1-\epsilon)\|x_i - x_j\|^2 \leq \|f(x_i) - f(x_j)\|^2 \leq (1+\epsilon)\|x_i - x_j\|^2.$$

There are $\binom{n}{2} \le n^2$ many pairs of data points in $(x_i)_{i=1}^n$ and hence

$$\begin{aligned}
&\mathbb{P}\left(\|f(x_i)-f(x_j)\|^2/\|x_i-x_j\|^2 \notin (1-\epsilon,1+\epsilon)\ \forall i,j \in [n]\right)\\
&\le \sum_{i,j\in[n],i<j} \mathbb{P}\left(\|f(x_i)-f(x_j)\|^2/\|x_i-x_j\|^2 \notin (1-\epsilon,1+\epsilon)\right)\\
&\le 2n^2 e^{-(\epsilon^2-\epsilon^3)p/4}.
\end{aligned}$$

Choosing $p \ge \frac{20}{\epsilon^2}\log n$ implies that

$$2n^2 e^{-(\epsilon^2-\epsilon^3)p/4} \le 2n^2 e^{-(1-\epsilon)5\log n} = 2e^{-(3-5\epsilon)\log n} < 2e^{-1/2\log n} \le 1$$

for $n \ge 4$, where we used that $\epsilon < 1/2$. Hence, the probability that f is an ϵ isometry for $(x_i)_{i=1}^n$ is not zero, which implies that such an f exists. $\square$

20.3 Diffusion maps

Principle component analysis as well as the Johnson-Lindenstrauss embedding compute a linear embedding into a lower dimensional space. Quite often, however, the intrinsic dimension is not adequately captured by a linear subspace. Consider, for example, Figure 29, where the Swiss roll data set is supposed to be mapped to a lower dimensional space. We see in the middle, that the projection approach via PCA fails to capture the intrinsic structure of the Swiss roll. On the right we find the method discussed in this section, which seems to do a much better job.

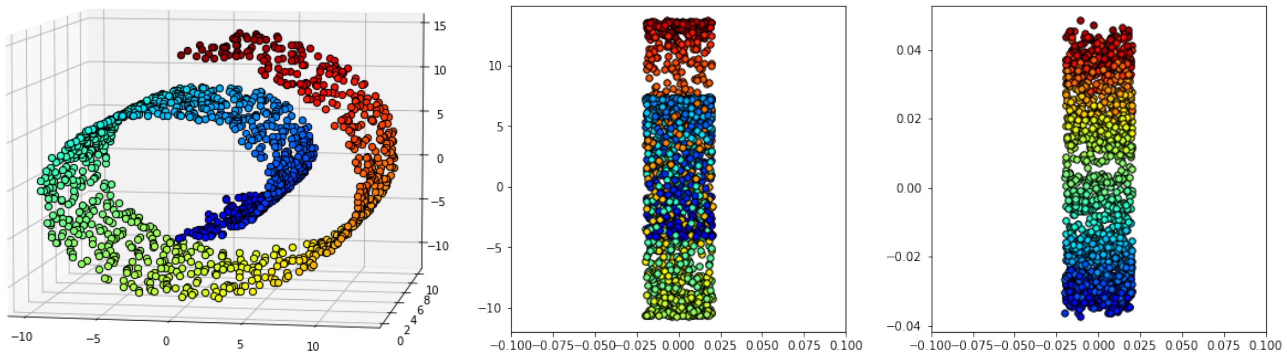


Figure 29: Swiss roll data set on the left. The middle figure shows an embedding resulting from projecting on a one dimensional subspace identified by PCA. On the right is the embedding obtained by the diffusion embedding discussed in this section.

It seems obvious that we do not want to maintain all pairwise distances in this nonlinear setting any longer. Instead we would rather like to have a map that in some sense respects the intrinsic distances of the points. To come up with a sensible notion of distance between points in a point cloud, that also respects the intrinsic structure, we consider the following example:

Example 20.1. *Consider the heat equation on a two dimensional domain: $u_0 \in L^2(\mathbb{R}^2)$, $\gamma > 0$*

$$\frac{\partial}{\partial t}u(t,x) - \gamma\Delta_x u(t,x) = 0 \textit{ for } (t,x) \in [0,T]\times\mathbb{R}^2 \tag{99}$$

$$u(0,\cdot) = u_0. \tag{100}$$

If u_0 is a small bump function centered at a point t_1 and $\tilde{u}_0$ is a second bump function centered at t_2 and both $\tilde{u}_0$ and u_0 have compact and disjoint support, then $\|\tilde{u}_0 - u_0\|_{L^2}^2 = \|\tilde{u}_0\|^2 + \|u_0\|_{L^2}^2$. From this information alone, we *obtain very little information about the distance of t_1 from t_2. However, if we instead look at $\|\tilde{u}(T,\cdot) - u(T,\cdot)\|_{L^2}$, where $\tilde{u}$ and u are the solutions of (99) with initial conditions $\tilde{u}_0$ and u_0 respectively, then these may give us a more nuanced description of the distance between t_1 and t_2. Consider Figure 30. There, we see that after some time has passed the distances between heat profiles are closely related to the distances between the initial heat sources.*

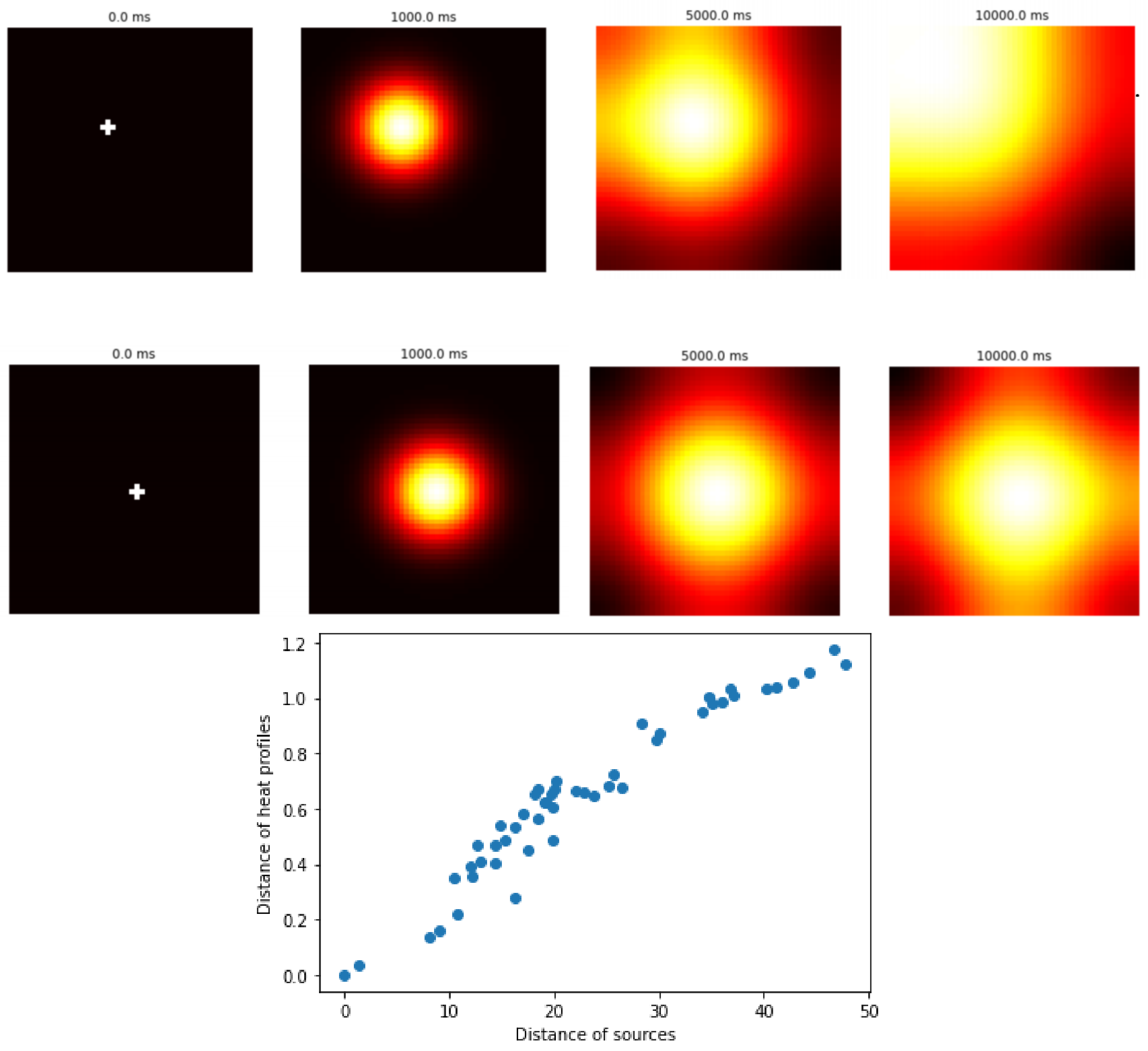


Figure 30: First and second row: Heat diffusion associated to two sources. Third row: relationship between distance of initial heat sources and heat profiles after two seconds of diffusion.

The interesting thing about the diffusion is that we can also use it to make sense of a distance on a non-euclidean domain.

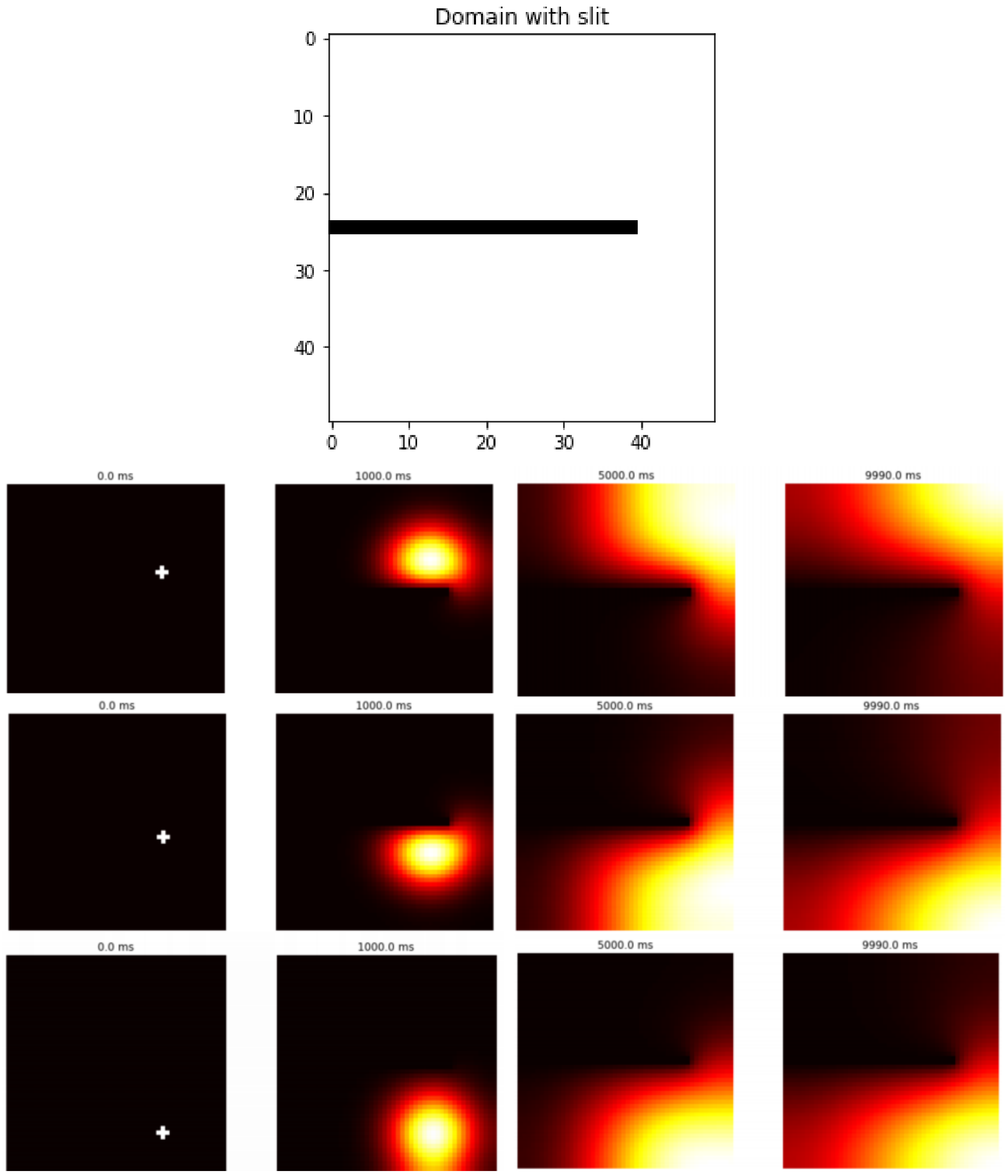


Figure 31: Diffusion on a map with a slit. The heat profile after two seconds now describes quite accurately the distance of points when taking into account the more involved geometry. Indeed, the two lower rows have a very similar heat profile after two seconds since the associated heat sources lie on the same side of the slit.

Example 20.1 showed us that, if we define a distance via the heat equation then this will oftentimes respect the underlying geometry. We will now do something similar for general graphs. Let (V, E, W) be a weighted graph. We define a random walk on V by

$$\mathbb{P}(X(t+1) = v_j | X(t) = v_i) = w_{i,j}/\deg(i). \tag{101}$$

For the graph corresponding to the slit domain of Figure 31, where every pixel in the white part of the image is one vertex of the graph and vertices corresponding to neighbouring pixels are connected by an edge with weight 1, we run some examples of the random walk in Figure 32.

We denote the matrix of transition probabilities by M, where $M_{i,j} = w_{i,j}/\deg(i)$. Note that $M = D^{-1}W$, where D is a diagonal matrix with $D_{i,i} = \deg(i)$. If we start the random walk X in the node i, i.e., $X(0) = i$, then we can compute the probability that $X(t) = j$ by

$$\mathbb{P}\left(X(t) = v_j | X(0) = v_i\right) = (M^t)_{i,j}.$$

As a result, we have that

$$\left(\mathbb{P}\left(X(t) = v_j | X(0) = v_i\right)\right)_{j=1}^{n} = e_i^T(M^t) = M^t(i,\cdot) \in \mathbb{R}^n. \tag{102}$$

The map $v_i \mapsto M^t(i,\cdot)$ can now be considered as a map from V to $\mathbb{R}^n$. This map now maps points to the associated probability distribution of the random walk starting in that point after t iterations. While we had observed that this embedding seems to reflect the inner geometry of the problem, it is hardly a dimensionality reduction since n may be quite large.

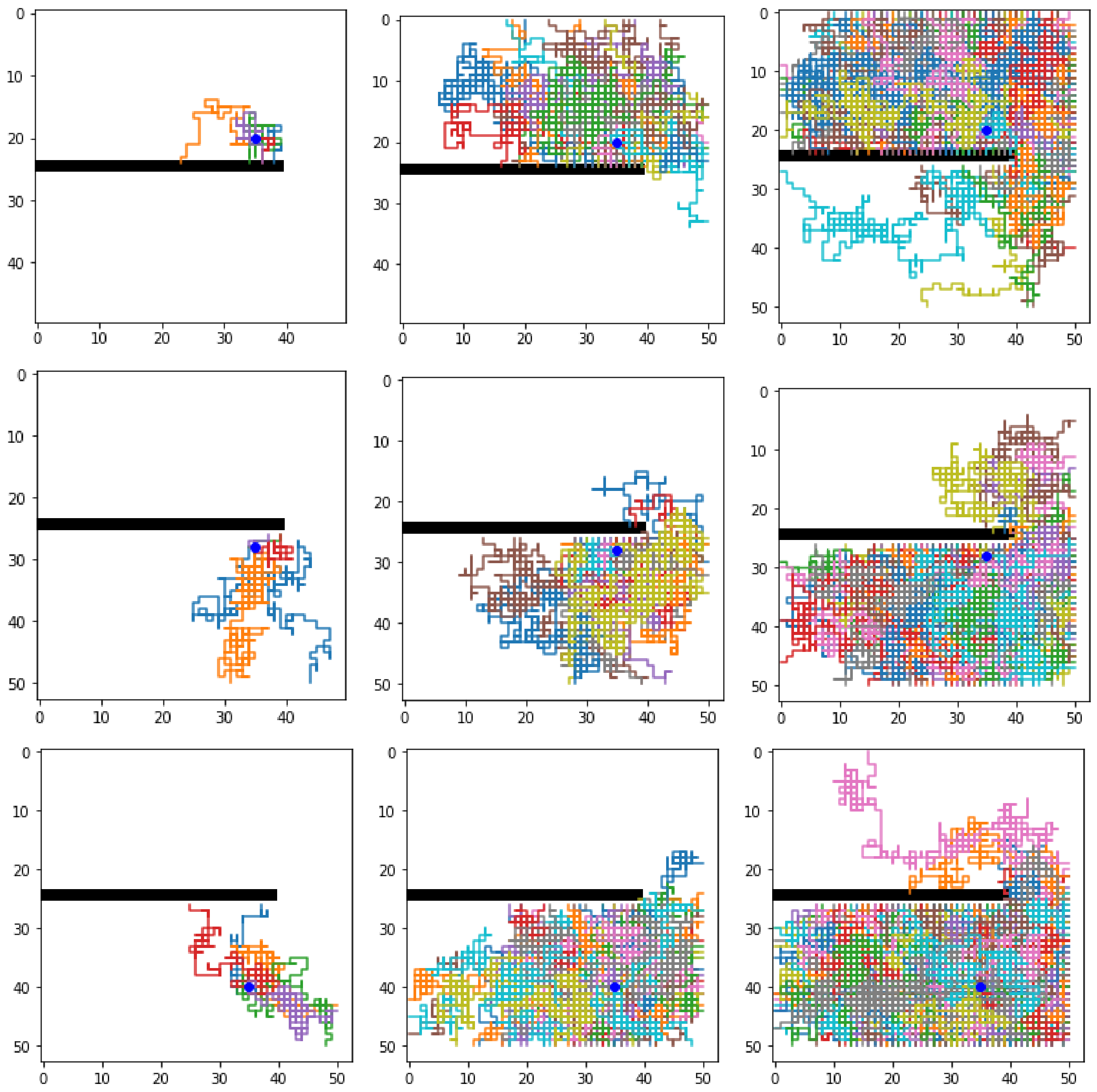


Figure 32: Three random walks starting at the same points as the heat sources of Figure 31. The points are marked by a blue dot in the images. The resulting distribution is very similar to the heat profiles of Figure 31.

To reduce the dimension of the embedding, we truncate the size of the matrix M by a spectral decomposition. First of all, we notice that

$$S = D^{1/2} M D^{-1/2} = D^{-1/2} W D^{-1/2}$$

is a symmetric matrix and therefore is equivalent to a diagonal matrix like

$$S = V \Lambda V^T,$$

for a matrix V such that $V^T V = \text{Id}$ and a diagonal matrix Λ with $\Lambda_{i,i} \geq \Lambda_{i+1,i+1}$ for all $i \in [n-1]$. Now we have that

$$M = D^{-1/2} S D^{1/2} = (D^{-1/2} V) \Lambda (V^T D^{1/2}) =: \Phi \Lambda \Psi^T.$$

Because of this, we can write

$$M = \sum_{k=1}^{n} \lambda_k \phi_k \psi_k^T,$$

where ϕ_i and ψ_i are the i-th column of Φ and Ψ, respectively. Note also, that by construction $\Phi\Psi^T = \mathrm{Id}$ and hence

$$M^t = \Phi\Lambda^t\Psi^T = \sum_{k=1}^{n} \lambda_k^t \phi_k \psi_k^T.$$

For the map $v_i \mapsto M^t(i, :)$ we now have that

$$v_i \mapsto \sum_{k=1}^{n} \lambda_k^t \phi_k(i) \psi_k^T, \tag{103}$$

Before we turn this representation into the so-called *diffusion map*, we observe that $\phi_1 = 1$. Indeed, it holds that $M1 = 1$. Therefore, $\phi_1 = 1$ holds if 1 is the largest eigenvalue of M.

Proposition 20.1. *All eigenvalues of M are bounded by 1 in absolute value.*

Proof. Let ϕ_k be a right eigenvector of M and let i_{max} be such that $\phi_k(i_{max}) = \max_{i\in[n]} \phi_k(i) > 0$. Then

$$\lambda_k \phi_k(i_{max}) = M\phi_k(i_{max}) = \sum_{j=1}^{n} M_{i_{max},j} \phi_k(j).$$

This implies that

$$\lambda_k = \sum_{j=1}^{n} M_{i_{max},j} \frac{\phi_k(j)}{\phi_k(i_{max})} \leq \sum_{j=1}^{n} M_{i_{max},j} = 1.$$

Similarly, we obtain that

$$-\lambda_k = \sum_{j=1}^{n} (-M_{i_{max},j}) \frac{\phi_k(j)}{\phi_k(i_{max})} \geq \sum_{j=1}^{n} (-M_{i_{max}}),_j = -1.$$

□

By this proposition and the previous discussion we conclude that $\phi_1 = 1$ and hence does not carry any information about G.

Definition 20.2. *Given a graph $G = (V, E, W)$ let $M = \Phi\Lambda\Psi^T$ be as above. For $t \in \mathbb{N}$, the* diffusion map φ_t *is defined as*

$$\varphi_t \colon V \to \mathbb{R}^{n-1}$$

$$\varphi_t(v_i) = \begin{bmatrix} \lambda_2^t \phi_2(i) \\ \lambda_3^t \phi_3(i) \\ \vdots \\ \lambda_n^t \phi_n(i) \end{bmatrix}.$$

The diffusion map is still not really performing dimensionality reduction. However, if we assume that many eigenvalues are significantly smaller than 1, then for sufficiently large t the values λ_k^t will be very small. Hence, we believe that we can drop the associated dimensions from the embedding without significantly affecting the embedding's quality.

Definition 20.3. *Given a graph $G = (V, E, W)$ let $M = \Phi\Lambda\Psi^T$ be as above. For $p \in [n-1]$ and $t \in \mathbb{N}$, the* truncated diffusion map $\varphi_t^{(p)}$ *is defined as*

$$\varphi_t^{(p)} : V \to \mathbb{R}^p$$
$$\varphi_t(v_i) = \begin{bmatrix} \lambda_2^t \phi_2(i) \\ \lambda_3^t \phi_3(i) \\ \vdots \\ \lambda_{p+1}^t \phi_{p+1}(i) \end{bmatrix}.$$

Let us conclude this section by proving that the diffusion map does indeed what we wanted it to do, which is to produce an embedding where the distances correspond to the distances of the probability densities of a random walk.

Proposition 20.2. *Let $G = (V, E, W)$ be a weighted graph, let $M = \Phi\Lambda\Psi^T$ be as above, and let X be a random walk as above. For every pair of nodes $v_i, v_j \in V$ and for every $t \in \mathbb{N}$ it holds that*

$$\|\varphi_t(v_i) - \varphi_t(v_j)\|^2 = \sum_{k=1}^n \frac{1}{\deg(k)} \left(\mathbb{P}(X(t) = k | X(0) = i) - \mathbb{P}(X(t) = k | X(0) = j)\right)^2.$$

Proof. By (102), it follows that

$$\begin{aligned}
&\sum_{k=1}^n \frac{1}{\deg(k)} \left(\mathbb{P}(X(t) = k | X(0) = i) - \mathbb{P}(X(t) = k | X(0) = j)\right)^2 \\
&\quad = \sum_{k=1}^n \frac{1}{\deg(k)} \left(\sum_{\ell=1}^n \lambda_\ell^t \phi_\ell(i) \psi_\ell^T(k) - \sum_{\ell=1}^n \lambda_\ell^t \phi_\ell(j) \psi_\ell^T(k) \right)^2 \\
&\quad = \sum_{k=1}^n \frac{1}{\deg(k)} \left(\sum_{\ell=1}^n \lambda_\ell^t (\phi_\ell(i) - \phi_\ell(j)) \psi_\ell^T(k) \right)^2 \\
&\quad = \sum_{k=1}^n \left(\sum_{\ell=1}^n \lambda_\ell^t (\phi_\ell(i) - \phi_\ell(j)) \frac{\psi_\ell^T(k)}{\sqrt{\deg(k)}} \right)^2 \\
&\quad = \sum_{k=1}^n \left(\sum_{\ell=1}^n \lambda_\ell^t (\phi_\ell(i) - \phi_\ell(j)) D^{-1/2} \psi_\ell^T(k) \right)^2 \\
&\quad = \left\| \sum_{\ell=1}^n \lambda_\ell^t (\phi_\ell(i) - \phi_\ell(j)) D^{-1/2} \psi_\ell^T \right\|^2.
\end{aligned}$$

Since $D^{-1/2}\psi_\ell^T = v_\ell$ per construction, where $(v_\ell)_{\ell=1}^n$ denote the eigenvectors of $S = D^{1/2} M D^{-1/2}$. Since $(v_\ell)_{\ell=1}^n$ form an orthogonal basis, we obtain that

$$\left\| \sum_{\ell=1}^n \lambda_\ell^t (\phi_\ell(i) - \phi_\ell(j)) D^{-1/2} \psi_\ell^T \right\|^2 = \sum_{\ell=1}^n \left(\lambda_\ell^t (\phi_\ell(i) - \phi_\ell(j))\right)^2 = \|\varphi_t(v_i) - \varphi_t(v_j)\|^2,$$

by Parseval's identity. $\square$

Example 20.2. *Using the diffusion map, we can produce a visual representation of data from which we only know the relationship between each pair of elements. For example, consider the situation that we only know which central European country as which neighbours, as given in Figure 33.*

	DEU	AUT	CHE	FRA	POL	CZE	ITA	DNK	SVN	NLD	BEL	SVK	HUN	HRV	LUX	LIE
Germany	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0
Austria	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0
Switzerland	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
France	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
Poland	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Czech Republic	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Italy	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Denmark	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Slovenia	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
Netherlands	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
Belgium	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
Slovakia	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
Hungary	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0
Croatia	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
Luxemburg	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
Lichtenstein	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 33: Adjacency matrix of countries in Europe. Countries that have a common border are considered to be connected by an edge.

From the local neighbourhood relationship of the countries, we cannot really see how central Europe looks like. We can, however apply a truncated diffusion map to the graph associated to that adjacency matrix and obtain the following embedding of Figure 34 two dimensional space.

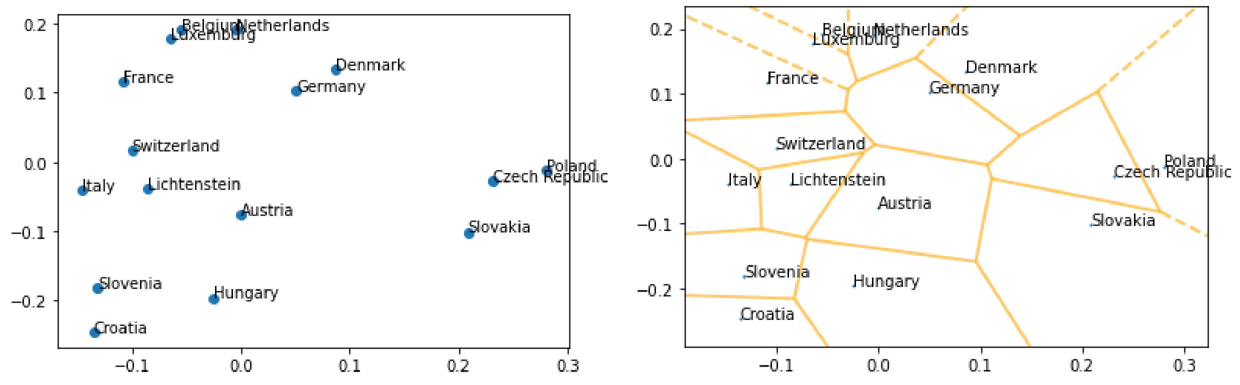


Figure 34: Embedding of the graph of Figure 33 in the two-dimensional plane. The positions coincide somewhat with the locations of the countries on a real map. On the right, Voronoi regions are drawn. They do not yield the correct borders. This is not surprising since Voronoi regions are necessarily convex, but countries are not.

References

[1] Peter L Bartlett, Vitaly Maiorov, and Ron Meir. Almost linear VC-dimension bounds for piecewise polynomial networks. *Neural computation*, 10(8):2159–2173, 1998.

[2] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[3] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[4] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[5] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.