HW1: Data Exploration and Preparation

Goal

This exercise is the first of three mini-projects to help you stop the spread of disease globally. We present the **Virus Test Challenge Dataset (VTC)** with labeled patient data. Your goal is to understand and prepare this dataset for prediction. While we will soon learn several methods for training prediction models, none of them will work without us first understanding and cleaning our data. So, in this exercise, we will perform standard data preparation practices to identify regularities and irregularities in the data.

Good Luck!

Source: xkcd

Instructions

- **Submission**
  - Submit by Tuesday, 02.07.24 (23:59) in groups of 2 students.
  - Submitted to the webcourse.

- **Python environments and more**
  - We recommend using jupyter notebooks. Google colab can be very convenient since it does not require installing anything on your local computer. It will also help you to collaborate with your partner online.
  - Initial notebook here.
    - Demonstrates how to upload a dataset to Google colab and how to download files from Google colab.
    - You can save a copy of this notebook to your Google drive.
  - However, you are allowed to use any Python IDE you choose. For working locally with an IDE, we recommend first installing conda for package management (with Python 3.6 or 3.8), and then installing an IDE like PyCharm or Spyder.

- **Your code**
  - Should be clearly and briefly documented.
  - Variables/classes/functions should have meaningful names.
  - May be partially reviewed and graded (בדיקה מדגמית).

- **Final report**
  - Should be written in a word processor (Office Word, Google docs, etc.).
    - Should not contain the code itself. Do not submit jupyter notebooks as PDFs.
  - Can be in Hebrew, English, or both.
  - **You are primarily assessed based on your written report.**
    - Will be partially reviewed and graded (בדיקה מדגמית).
  - Answer the questions in this instruction file according to their numbering.
  - Add concise explanations, figures (outputs of your code), tables, etc.
  - Tables should include feature names and suitable titles.
  - Plots:
    - Must be clear, readable, and coherent.
    - Should have suitable titles, axis labels, and legends (if needed).

        - Should have grid lines (except maybe heatmaps).
        - We recommend adjusting the default font sizes of matplotlib at the beginning of your notebook. You can use the following code snippet:

```
from matplotlib import pylab
params = {'xtick.labelsize': 18,
          'ytick.labelsize': 18,
          'axes.titlesize' : 22,
          'axes.labelsize' : 20,
          'legend.fontsize': 18,
          'legend.title_fontsize': 22,
          'figure.titlesize': 24 }
pylab.rcParams.update(params)
```

    - You are evaluated for your answers but also for readability, clarity, and aesthetics.

- **Submit a zip file containing** (please use hyphens, not underscores):
    - Define *<filename>* as your dash-separated IDs, i.e., *id1-id2* or *id1-id2-id3*.
    - The zip file's name should be *<filename>.zip* (e.g., *123456789-200002211.zip*).
    - **Only one group member should submit the assignment to the webcourse!**
    - The report PDF file with all your answers (but not your code!), named *<filename>.pdf*.
    - Your code (choose the relevant options for you):
        - Working with jupyter: your notebook, *<filename>.ipynb*.
        - Your completed kNN module (=class) from Part 2, *kNN.py*
        - Working with a "traditional" IDE: one clear main script, *<filename>.py*, and any additional files required for running the main script.
        - Data preparation function *prepare.py* (from Part 6).
    - Do not submit csv files.
- **Failing to follow any of the instructions above will lead to point deduction!**

Part 1: Data Loading and First Look

The VTC dataset, available on the course website as `virus_data.csv`, should be loaded and explored using the pandas library. It contains features relevant to our prediction tasks, along with ground-truth labels for our target variables: **`spread`** (potential to spread COVID-19) and **`risk`** (risk of serious illness). All your decisions in the data preparation process should be made with these targets in mind.

Unfortunately, as with any real-world dataset, VTC includes many redundancies and noise. Throughout this exercise, we will work to minimize these issues. Note: The dataset is synthetic and may not match real-world statistics.

(Q1) Load the dataset into a Pandas `DataFrame`.

Answer (in your report)**:** how many rows and columns are in the dataset?

1250 rows and 25 columns.

Before we continue, let us define the "ordinal" variable type. Ordinal variables are categorical with a natural order (e.g., year of birth), and are somewhere between continuous and categorical variables.

(Q2) Print the `value_counts` of the `conversations_per_day` feature (see Tutorial 01).

Copy the obtained output to your report. Describe in one short sentence what you think this feature refers to in the real world.

This feature's type is "ordinal". Explain briefly why.

Remember to clearly write the number of the question next to your answer.

Output:

```
conversations_per_day
3     218
2     204
5     179
4     168
1     108
6     107
7      94
8      54
9      42
10     29
11     16
13      8
12      7
14      6
16      5
15      3
17      1
29      1
Name: count, dtype: int64
```

We think that the feature 'conversations_per_day' expresses the number of different people that a patient has met per day on average. And the 'value_counts' of the 'conversations_per_day' feature expresses the amount of patients that have met the same number of people per day.

For example, the first row represents 218 patients that have met 3 people per day on average.

This feature is an ordinal variable because it's categories are conversation per day and have the natural order on it (decreasing).

(Q3) In your report, write a table describing each feature. The columns must be:

a. Feature name: the name of the feature as it is written in the dataset.
b. Description: a short sentence with your understanding of the feature's meaning in the real world.
c. Type: Continuous, Categorical, Ordinal, or Other.
   Don't overthink this (especially the "ordinal" type), some variable may be suitable for two types.

Note: do not include the target columns ("`spread`" and "`risk`").

Feature name	age	sex	weight	blood_type	current_location	num_of_siblings	happiness_score	household_income	conversations_per_day	sugar_levels	sport_activity
Description	patient age	patient sex	patient weight	patient group of blood	patient location	number of sibling the patient has	patient level of happiness	patient economical status	number of conversation patient makes per day	patient sugar rate	level of sport activity patient does
Type	continuous	Categorical	continuous	Categorical	other	Categorical	Ordinal	Ordinal	Ordinal	Ordinal	Categorical

Feature name	pcr_date	PCR_01	PCR_02	PCR_03	PCR_04	PCR_05	PCR_06	PCR_07	PCR_08	PCR_09	PCR_10
Description	the date the pcr tests were taken	PCR_01 test results	PCR_02 test results	PCR_03 test results	PCR_04 test results	PCR_05 test results	PCR_06 test results	PCR_07 test results	PCR_08 test results	PCR_09 test results	PCR_10 test results
Type	other	continuous	continuous	continuous	continuous	continuous	continuous	continuous	continuous	continuous	continuous

Partitioning the data

During the learning process, we measure our models' performance on two disjoint sets: **training** and **test**. A training set is a subset of the dataset from which the machine learning algorithm learns relationships between features and target variables. The test set provides a final estimate of the machine learning model's performance after it has been trained. Test sets should never be used to make decisions about which algorithms to use or for improving or tuning algorithms.

We will explore why this data partitioning is important later in the course, but for now, the most important thing to remember is that **you may only use the training set** for making decisions about the data, training our models, plotting graphs regarding our data, and deciding how to normalize features. The training set will help us understand what pre-processing steps we need to use on the data. **You should then apply those pre-processing steps to both the training and the test set.**
Note: later in the course, we will use another data subset, called the validation set.

(Q4) Split the data randomly into a training set (80% of the data) and a test set (20% of the data). As the `random_state`, use the sum of the last two digits of each of your IDs[1] (two or three IDs).
The random state will ensure that you get the same split every time.
Answer: Why is it important that we use the exact same split for all our analyses?

Answer:
By using the same random state we can ensure that for each split we get the same train and test data sets.
It is important that we use the exact same split for all our analysis because if we will change the training data overall the model has learned which could lead to overestimating or underestimating the model accuracy.
Also having a consistent split allows for a fair comparison. If the data was split differently each time, differences in results could be due to the different data rather than the efficacy of the models themselves.

Note: it could be easier for you to answer this question after you complete the rest of the assignment.

[1] i.e., if my i.d is 200033035 and my partner's is 300011016, then the random state should be 35+16=51.

Part 2: Missing Values

We will start with basic checks of our datasets, focusing on missing values, which are a common issue in machine learning. Missing values refer to data points that are absent in a specific column, often represented as null values. They pose a significant challenge in data analysis and can lead to inaccurate or biased results. Data can be missing due to technical issues, human errors, privacy concerns, and more.

(Q5) For **both the training set and test set**, report which fields have missing values and how many missing values there are. You can use Panda's function isnull().

Both in training and test set, the only value missing is for `household_income`.

Number of missing value:

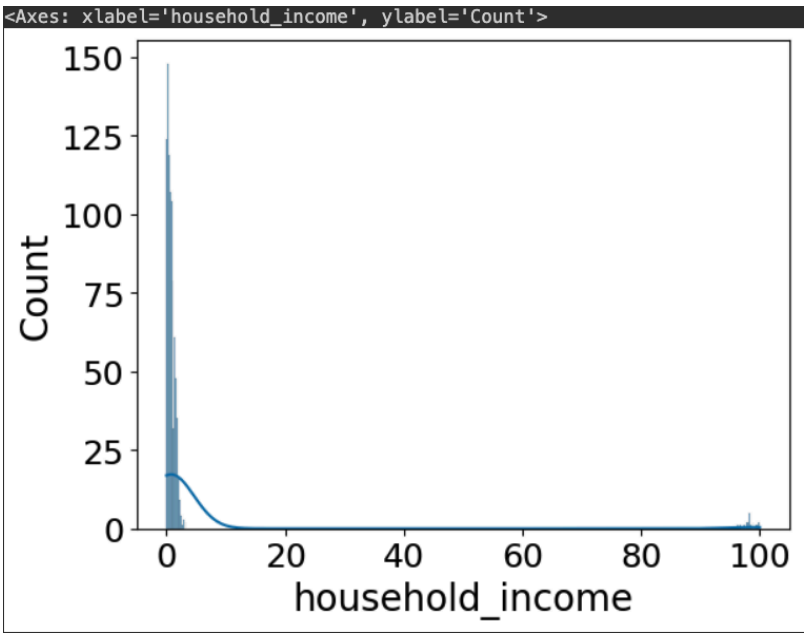
training set: 77

test set: 32

Another term we will use is "Outliers". There are many ways to define outliers; here, we refer to them as data points that significantly deviate from the rest of the data. Note that we won't solve the outliers issue in this section, only the missing values issue.

(Q6) Plot a histogram (see Tutorial 01) for each field where you found missing values in **(Q5)**. Add these plots to your report. Answer: Can you recognize outliers?

Reminder: Create plots using only the training set.



The outliers are the few patients with hosehold_income high, close to 100.

Most of the data is between 0 to 5.

There are many ways of dealing with missing values. We will consider two of them:

a. Calculate this field's **mean** value in the train set and use it to replace the missing values in both the train and test set.
b. Like (a), but using the **median** instead of the mean.

(Q7) For each field where you found missing values, calculate the median and the mean in the training set, **and report it**.
If there is a significant difference between the mean and median values, explain the reason. Which filling method do you prefer to use in our case, and why?

Mean value in training set: 3.099133
Median value in training set: 0.700000

The mean is about 4 times greater than the median. This is due to the outliers causing high variance.
Thus, in our case filling the missing value with the median would represent real life more accurately.

Task A: Use the method you chose to fill the missing values in **both training and test sets**. You can use Panda's function fillna().

Part 3: Warming up with k-Nearest Neighbors

In this part, we focus on the `spread` target variable and start with one of the simplest models we know, "k-Nearest Neighbors".

Reminder: we use only the training set for now.

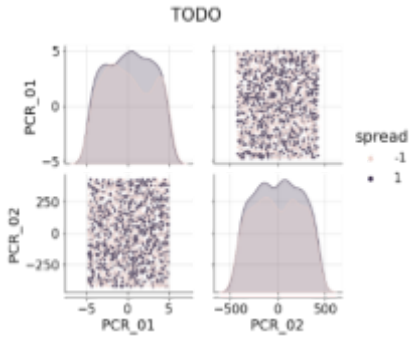
Basic data exploration

Our medical experts suspect that it is possible to predict the `spread` using a pair of `PCR` features from the set: {`PCR_04` , `PCR_07` , `PCR_09`}.

Task B: For each possible pair from this set, create a `seaborn.pairplot` of the two aforementioned `PCR` features. Use the `hue` parameter to color the different (train) data points according to their `spread`. Do not attach all these figures in your report.

Following is a code snippet that can help you start, and an example of the resulting figure (with different data):

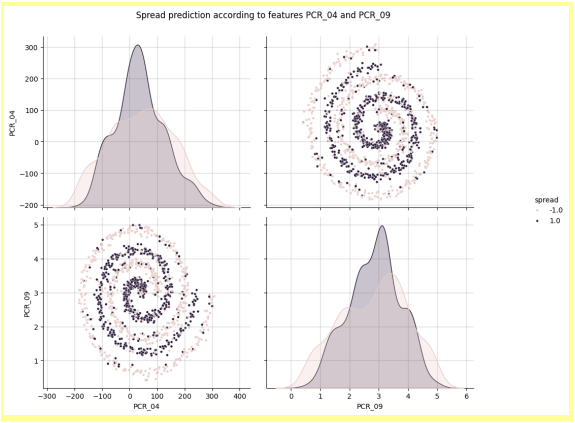
```
g=sns.pairplot(TODO, plot_kws={"s": 12})
g.fig.suptitle("TODO", y=1.04)

for ax in np.ravel(g.axes):
  ax.grid(alpha=0.5)

g.fig.set_size_inches(12,8)
```



(Q8) Answer briefly: Based on the plots you created on **(Task B)**, what pair of features is useful for predicting the `spread`?

Attach the `seaborn.pairplot` of only this pair of features to the report. Make sure your plots are readable and clear, and that they have proper titles, grid lines, axis labels, etc. Any missing requirement will lead to a points deduction!

The pair of features PCR_04 and PCR_09 is the most useful for predicting the `spread`. In this pair, spread labels are most "separable" out of all other pairs.



k-NN implementation

Our first step is to implement a basic k-NN classifier. We will inherit the `BaseEstimator` class from sklearn for compatibility with scikit-learn API. We will also inherit `ClassifierMixin` which will automatically add accuracy scoring function to our model.

Task C: Implement k-NN using the code template below (don't change method signatures):

```
from sklearn.base import BaseEstimator,ClassifierMixin

class kNN(BaseEstimator, ClassifierMixin):
    def __init__(self, n_neighbors:int = 3):
        self.n_neighbors = n_neighbors

    def fit(self, X, y):
        # TODO: complete
         return self

    def predict(self, X):
        # Note: You can use self.n_neighbors here
        predictions = None
         # TODO: compute the predicted labels (+1 or -1)
         return predictions
```

Avoid using `for` loops, `list`, `map`, `lambda`, etc.

Tip: Read about `scipy...cdist`, `np.copy`, `np.argsort` (or better: `np.argpartition` [1],[2]).

(Q9) What is the time complexity of the prediction function you wrote, applied on a single test datapoint, in terms of the number of neighbors k, the number of training datapoints m and the data dimension d? Explain. It is okay to "estimate" the complexity of python library functions. For instance, if you use `np.argsort` on n elements, then its complexity should be $O(n \log log\, n)$. Use your reason and CS knowledge.

Answer:

Time complexity to find distances using *cdist*:

calculating distances for a single point in d dimensional data (2) from each of the m (1000) training points is O(md).

Time complexity to find closest k neighbors indexes using *argpartition*:

argpartition time complexity is linear to its input size.

For a single test point, argpartition finds its closest k-neighbors in O(m).

Time complexity to determine neighbors labels using *array selection*:

For a single test point, and given k indexes, we select the values at the indexes in O(k).

Time complexity to determine most common label using *Counter from collections*:

For a single test point, counting frequency of k labels is O(k) .

Time complexity to set predictions using *apply_along_axis*:

For a single test point, we would apply most_common_labal only once, thus this part will take O(k).

All together, we have a time complexity of O(md) + O(m) + 3O(k).

Since k could be at most of size m and md is greater or equal to m, one has time complexity of O(md).

We will now test your implementation.

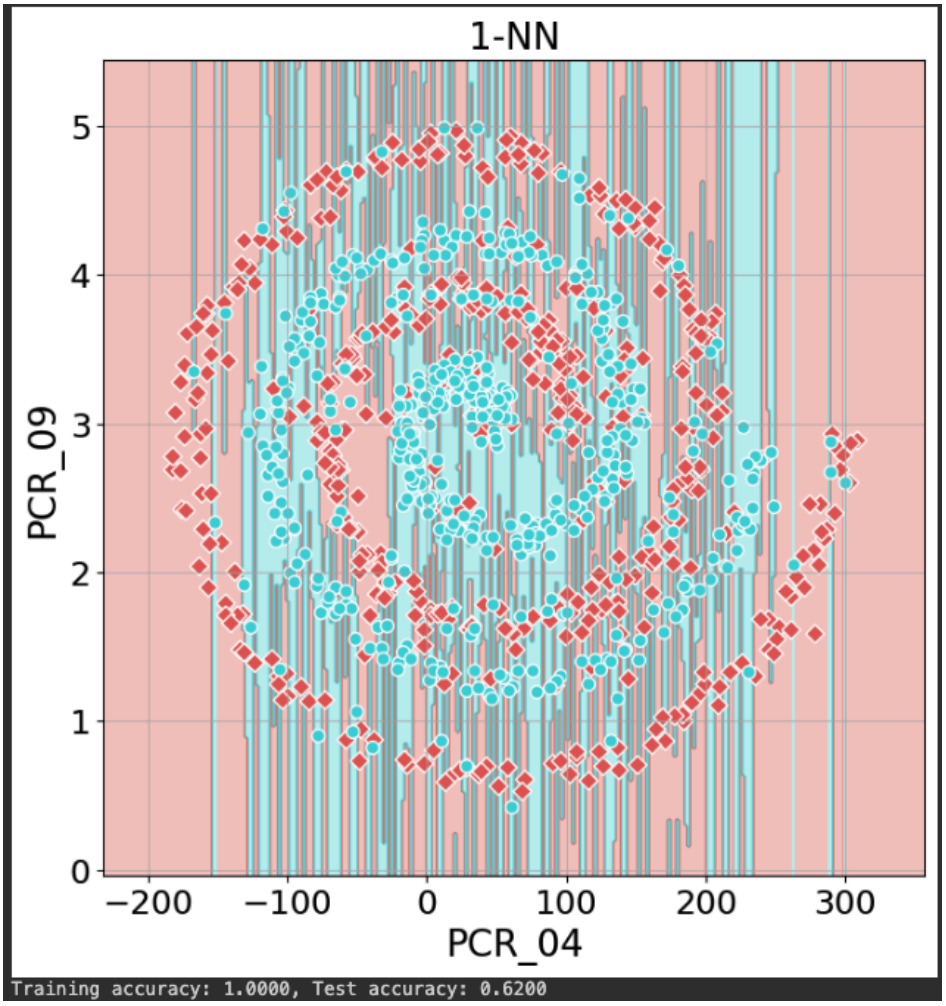
Task D: Create a temporary `DataFrame` by taking only the two features you chose on **(Q8)** from the training set. Train a `1-NN` model (with `k=1`) on this subset to fit the `spread` label. Use the provided `visualize_clf` function to visualize the decision regions of the model (send only the training set to this function, so that only the training examples will be scattered on the plot).

Compute the training accuracy and test accuracy of the model by calling its `score` method, e.g., call `h.score(Xtrain, Ytrain)`.

Make sure that all labels in your notebook (the ones in the dataset and the ones your model return) are ± 1, and not $\{0, 1\}$ or $\{True, False\}$.

(Q10) Attach the figure to your report. Specify the model's training and test accuracies.

(The plot should exhibit a bizarre behavior which we will discuss next.)



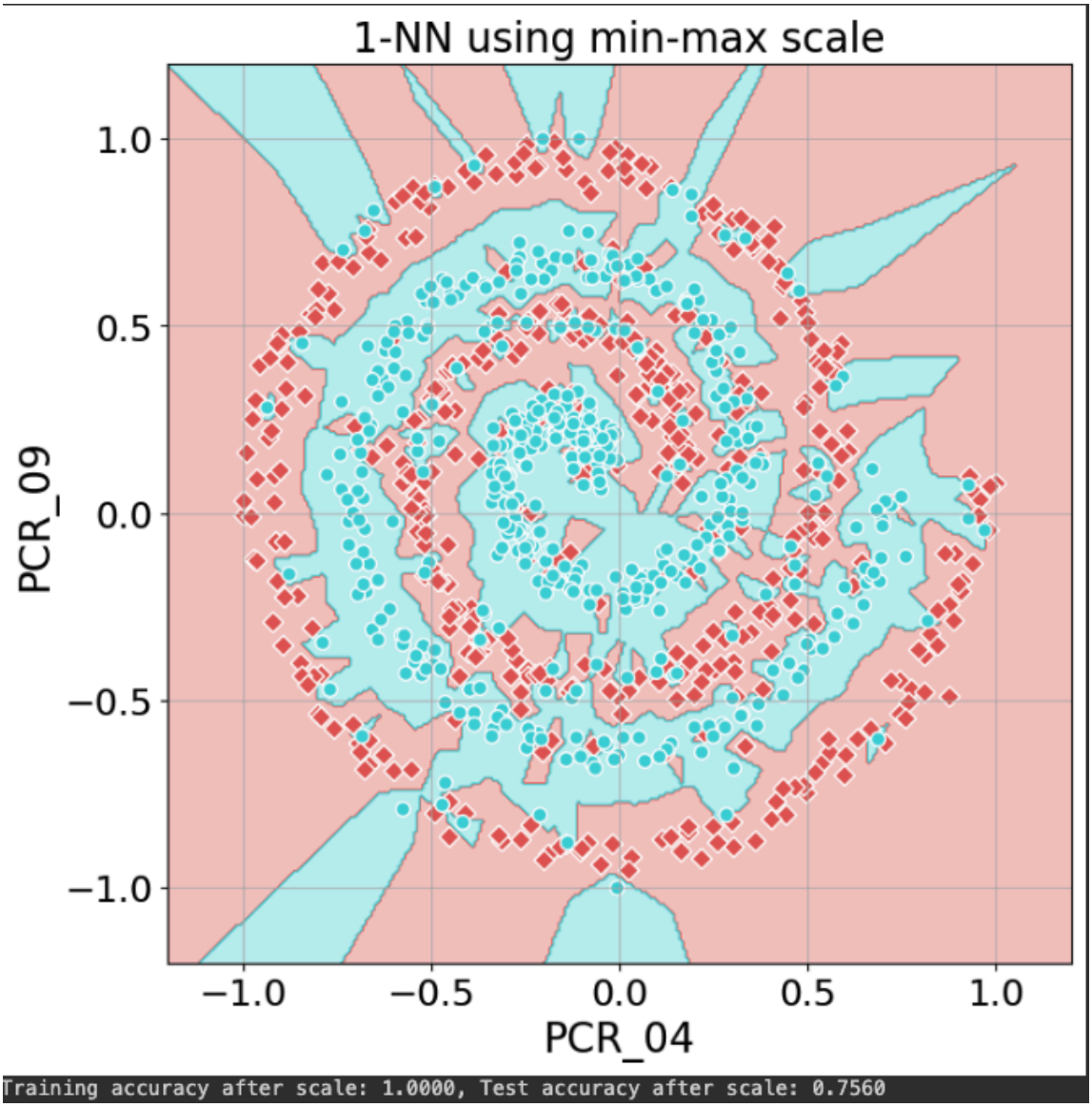
Data Normalization

In machine learning, it is common to normalize the features, as this allows us to develop better models (we will see soon why). We now focus on two normalization techniques: Standardization (Z-score) and min-max scaling (read the explanations in the links). Implementations can be found here and here.

Notice: we often ask questions in the exams regarding reading materials that appear in the assignments.

(Q11) Use min-max scaling (between $[-1, 1]$) to normalize the two features in the temporary `DataFrame` you created before, and train a new kNN model ($k = 1$) on the normalized dataset.

Compute the new training and test accuracies and draw the decision regions of the model. Attach the results to your report and compare them to those from **(Q10)** for the same $k = 1$ model on the raw data. Use these results to explain why normalization is important for nearest neighbor models.



It can be obtained that before normalization, the test accuracy was 0.6200 meanwhile after min-max

scaling we improved the test accuracy to 0.7560! This is around 20% improvement in our prediction

which is amazing.

Training accuracy remains the same.

We can infer that normalization is important for nearest neighbor models because KNN is basically based on the distance computation between the feature values. So, when some values are larger than others, those features of larger values will dominate the similarity and distance, leading to misleading performance, so to overcome this problem, we can bring down all the variables to the same scale.

(Q12) Using the normalized dataset, train another kNN model with $k = 5$. Compute the training and test accuracy and draw the decision regions of this model.

Attach the results to your report and compare them to those from **(Q11)**.

Use these results to briefly explain the effect of k on the decision regions.

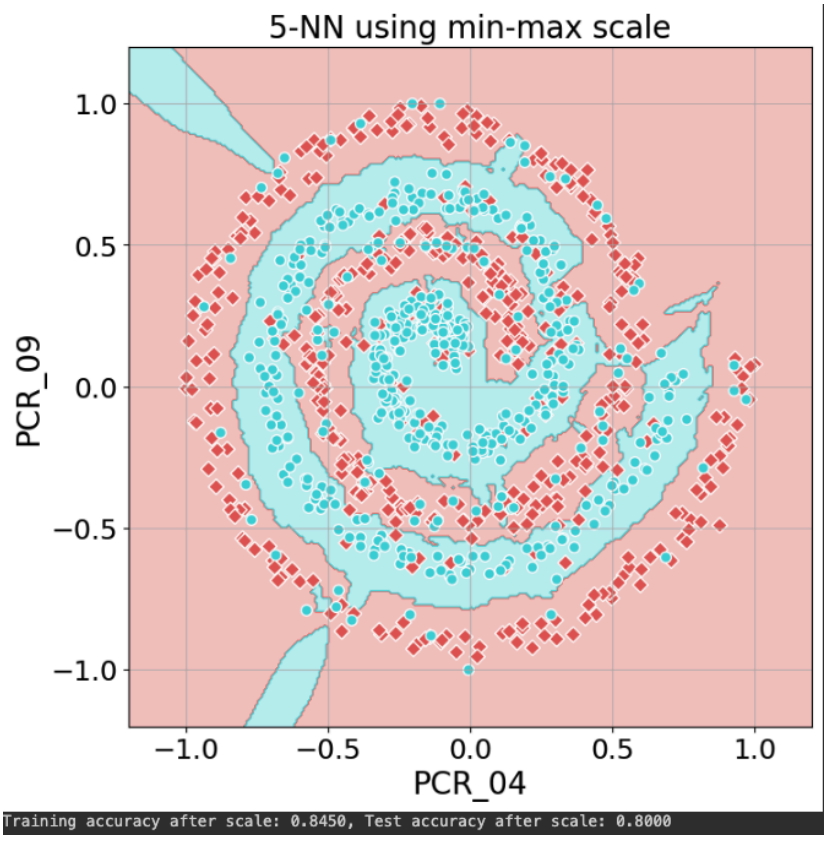
Answer:

In Q11 the train had 100% accuracy, because for k=1 each point is classified by itself, leading to perfect classification on the training set.

Training Accuracy for k=5: lower than k=1 because the model averages the classification over 5 neighbors, leading to some points being classified differently than their exact label.

High accuracy like that could cause overfitting in the test set and be very sensitive to each point in the training set. It can be noticed in the test accuracy which was 75%.

Now the train has 84% accuracy, because each point's label is set accordingly to its 5 nearest neighbors. In that way the variance decreases and dealing with the overfitting for better generalization, improving the accuracy of the prediction of the test set. It can be seen that test accuracy increased and its value was 80%.



(Q13) This question is general and does not deal with the given dataset. Assume a dataset with two features, one randomly sampled (i.i.d.) from a uniform continuous distribution on the range $[2, 5]$ and the other randomly sampled (i.i.d.) from a chi-squared distribution $\chi^2(k = 2)$ (see in Wikipedia).

(The labels are determined by some unknown function of these two features.)

Why is normalizing both features using min-max scaling to $[-1, 1]$ a bad idea? Explain in detail.

Answer:

Denote feature **X** = randomly sampled (i.i.d.) from a uniform continuous distribution on the range $[2, 5]$ and feature **Y** = randomly sampled (i.i.d.) from a chi-squared distribution $\chi^2(k = 2)$.

First of all, the way the Min-Max scaling works is by transforming the largest value to 1, smallest value to -1 and compressing the other data according to the new boundaries. Hence, this method of scaling is sensitive to outliers.

While X is evenly spread on the interval [2,5], applying Min-Max scale will work nicely, preserving the uniform distribution.

Although, Y has its most data on its lower boundary and a long tail at the end. This behavior could be obtained as outlier and after applying Min-Max scaling, the greatest value will be transformed to 1 and the other major data at the beginning will be compressed to a narrow interval in [-1,1] which could lead to lack of generalization and disturbing of the original distribution. All together, this kind of scale is not a good choice for Y.

So our conclusion is that Min-max scaling to $[-1, 1]$ is a bad idea because it doesn't account for the differences in the distributions of the features. It compresses the chi-squared feature's values, losing meaningful variance and potentially distorting the feature space.

Part 4: Data Exploration

We are now ready to start the preprocessing stage for the rest of the features!

Our medical experts suggest that blood types affect the `risk` target variable. They propose merging blood types into two groups: `{O+, B+}` and `{O-, A-, A+, B-, AB+, AB-}`. That is, instead of having a separate Boolean feature for each blood type, we would have one Boolean feature for these groups.

Task E: According to the suggested groups, create a new Boolean feature called `SpecialProperty` in your `DataFrame`, indicating whether the specific data point has a blood type in `{O+, B+}` or not. Then, remove the original `blood_type` feature from the `DataFrame`.

Technical: You can use the following snippet as a starting point to create a Boolean series according to a subset of the values of a feature:

```
df["blood_type"].isin(["O+", "B+"])
```

Univariate Analysis

You will now carry out most of the univariate analysis in your notebook (or IDE).
You should not add all the plots to the report, only the ones we specifically request.

For every **numerical** feature (including extracted ones), plot two histograms, one for each target variable (`risk` and `spread`), using hue to split by the target variable's value (e.g., high/low `spread` value). For continuous/ordinal features you should use the kde keyword to draw the estimated distribution curve (see Tutorial 01).

The following code snippet generates a 2-column figure of histograms of the features in the `COL_NAME` list. You may use this as a template to generate meaningful plots.
Refer to the seaborn documentation to understand more on `histplot`'s keyword arguments.

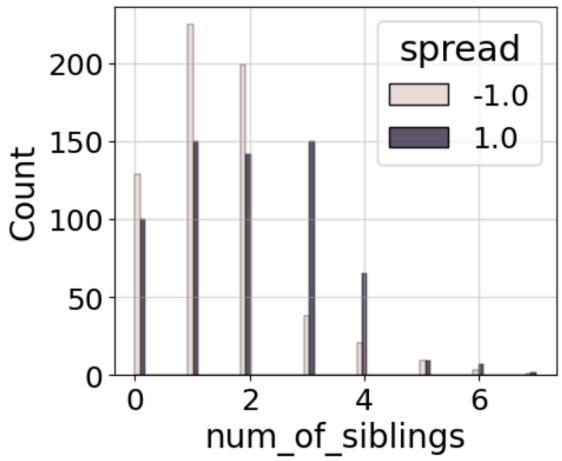
To clarify: in your jupyter notebook you should generate 2 histograms for every feature.
Each histogram corresponds to one target feature (`risk`, `spread`), where the different labels are counted separately and colored differently. Continuous variable histograms should also have estimated distribution curves (using the `kde` argument).

(Q14) According to the univariate analysis, name one feature that seems informative for predicting the `spread` target variable (other than the 2 features from **Q8**).

Attach the appropriate univariate plot and briefly explain (2-3 sentences) why this plot makes you think that feature is informative.

Answer:

The feature that seems informative for predicting the `spread` target variable is the *num_of_siblings* because in almost all the num_of_siblings categories, the value count has a big difference per spread value. Meaning we can use it to predict spread, given the number of siblings a patient has.

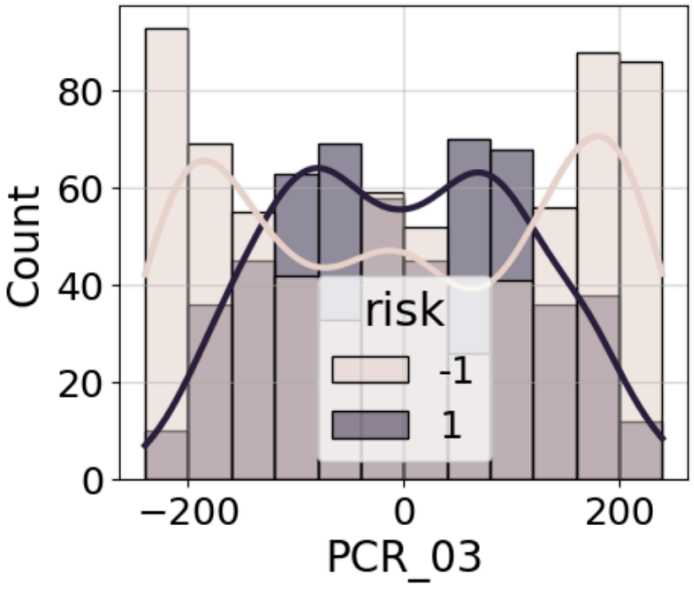


(Q15) According to the univariate analysis, name one feature that seems informative for predicting the `risk` target variable (other than the blood groups).

Attach the appropriate univariate plot and briefly explain (2-3 sentences) why this plot makes you think that feature is informative.

Answer:

The feature that seems informative for predicting the `risk` target variable is the feature *PCR_03* because it can be seen that for values in [-100,100] (approx) there are more risk cases than not, so it is more likely to be in risk (risk = 1), meanwhile for values in [-200,-100] or [100, 200] the risk sharply decreases.



Bivariate Analysis

We will now perform some bivariate analysis.

The following snippet performs basic bivariate analysis for the `PCR` features, conditioned on the `risk` variable. This snippet is a good reference for the next question.

```
sns.pairplot(df[df.filter(like='PCR').columns.tolist() + ["risk"]],
             plot_kws={"s": 3}, hue="risk")
```

Our medical experts believe that it is possible to predict the `risk` using a pair of PCR features from the set: **{`PCR_01`, `PCR_03`, `PCR_05`, `PCR_10`}**, **but only after splitting the data according to blood groups.**

(Q16) Split the (training) data based on the binary `SpecialProperty` feature created in **(Task E)**. For each split, perform a bivariate analysis for the PCR features in the set, in relation to the `risk`. This means you should produce two "matrices" of plots, one for each blood group. Each matrix should contain 4x4 subplots, representing all possible pairs of PCR features in the set. Do not include these plots in your report.

According to those plots, choose a pair of `PCR` features that could be helpful for predicting the `risk` (with the partition according to the `SpecialProperty`). What `PCR` features did you choose? And why?

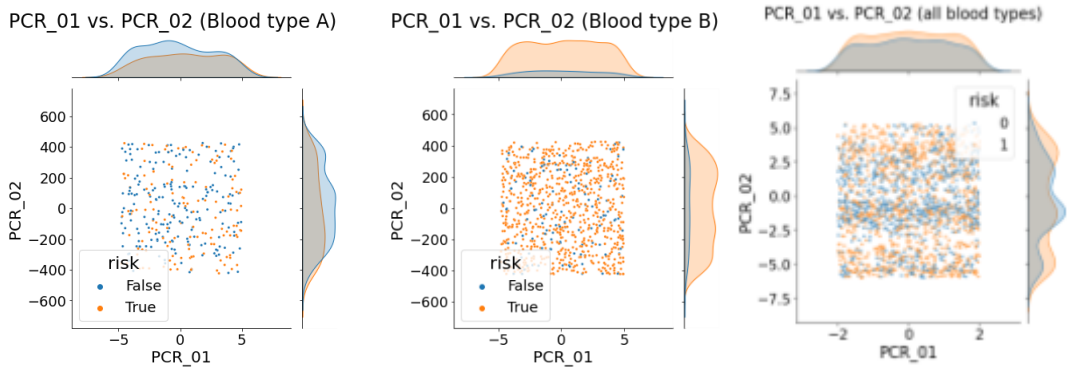
Answer:

The PCR features that we chose are `PCR_03` and `PCR_10`. In their plots we can see that there is some separability of the risk distribution.

(Q17) For the pair of `PCR` features you chose in **(Q16)**, create three `jointplot`s (see Tutorial 01), all conditioned on the `risk` variable. The first `jointplot` should include only the data in the first blood group you created in **(Task E)**, `{O+, B+}`. The second `jointplot` should include only the data in the other blood group. The third `jointplot` should be for the full data, without partitioning to blood groups.

Attach the 3 resulting plots to your report. Remember to have grids, titles, and axis-labels.

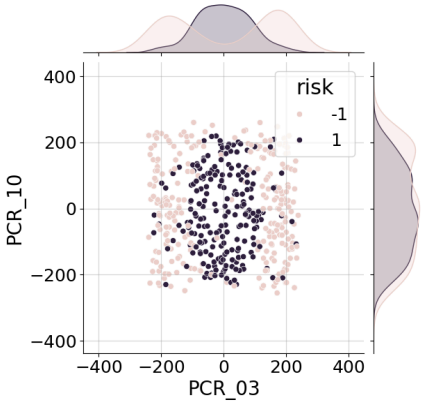
Example (for one hypothetical pair):



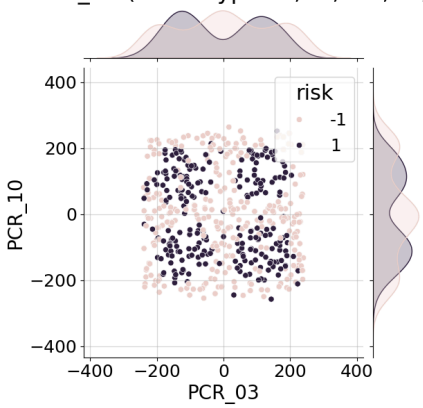
Notice that the three plots above here do not look informative for predicting the risk in any of the blood groups (look at the marginal distributions for instance).

Answer:

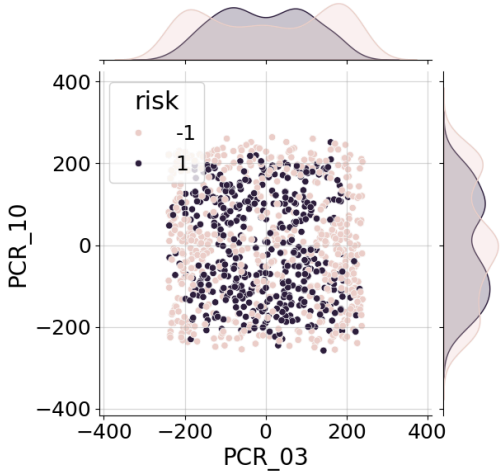
PCR_03 vs. PCR_10 (Blood type O+ , B+)



PCR_03 vs. PCR_10 (Blood type O-, A-, A+, B-, AB+, AB-)



PCR_03 vs. PCR_10 (all blood type types)



-

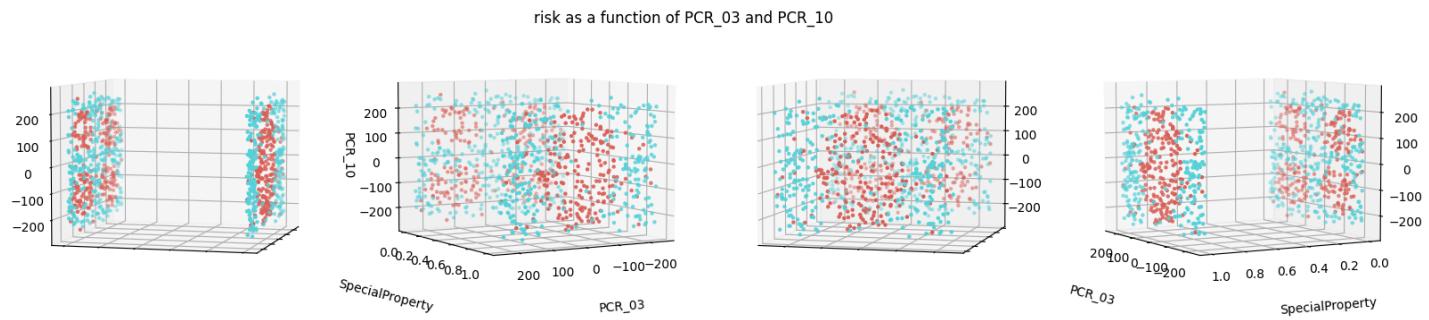
The `risk` target feature

The following snippet demonstrates a call to the provided `plot3d` function:

```
plot3d(df, "PCR_01", "SpecialProperty", "PCR_02", title="TODO", hue="risk", s=5)
```

(Q18) Use the provided function `plot3d` to plot the pair of `PCR` features you chose (axes X and Z) and the `SpecialProperty` feature (axis Y), colored by the `risk` label. Make sure that the plot is clear & readable and that it has a proper title. Attach the plot to your report.

Answer:



In the following questions, we will analyze how well various models fit the training data. We will use three features: the two `PCR` features and the `SpecialProperty` feature, aiming to fit the `risk`. Explain your answers in detail (2-5 sentences per model). No code execution is required. These questions focus only on whether a model can fit the training data, not on generalization (i.e., test data) or optimization (finding the model).

(Q19) How well will a decision tree of `max-depth=3` be able to fit the training data? Explain briefly.

Answer:

A decision tree of `max-depth=3` will be able to fit the training data not so well, it will capture the most dominant patterns in the training data but will likely underfit due to its inability to represent more complex relationships and interactions.

(Q20) How well will a decision tree of `max-depth=30` be able to fit the training data? Explain briefly.

Answer:

A decision tree with a max depth of 30 will be able to fit the training data very well, capturing the detailed patterns and variability present in the data. However, this is likely to lead to overfitting, where the model captures noise and does not generalize well to new, unseen data.

(Q21) How well will a `1-NN` model be able to fit the training data? Note that in this question, a point in the training set is not considered its own neighbor (i.e., when making a prediction for a training data point, the model won't use the same point for prediction, but only the nearest point in the remaining training set).

Hint: consider the scale of the features in your answer.

Answer:

A 1-NN model will fit the training data very well, because it uses the nearest neighbor for predictions. However, due to the different scales of the features, the model's performance might be biased toward features with larger ranges, such as PCR_03 and PCR_10.

Part 5: More Data Normalization

We will now complete the normalization process for all the `PCR` features.

Task F: Use the univariate analysis above to choose an appropriate normalization method (see Part 2) for every `PCR` feature in your `DataFrame`. Accordingly, apply sklearn's StandardScaler and MinMaxScaler to those features.

In **(Q23)** you are asked to specify the normalization method you chose for each feature.

Hint: Think about **(Q13)** when deciding which scaler to use for each feature.

(Q22) What will be the effects of data normalization on your answers in **(Q19)**, **(Q20)**, **(Q21)**?

Answer:

The only models that will be affected by normalization are models that predict based on distance. Thus, since decision trees don't rely on distance but kNN does, Q21 is the only one to be affected and the precision might be better.

Part 6: Data Preparation Pipeline

We have finished exploring and preparing our data. Throughout this assignment, you transformed features, normalized the data, and so on.

(Q23) Write a table summarizing the data preparation process you created.

The columns of the table must be:

a. **Feature name**: the name of the feature as written in the dataset.
   Names of new features should be meaningful!
b. **Keep**: "V" if the feature is kept, "X" otherwise (e.g., `blood_type` is removed).

c. **New**: “V” if the feature was handcrafted using other feature(s), “X” otherwise.

d. **Normalization method,** if used.

Feature name	**Keep**	**New**	**Normalization method**
patient id	X	X	-
age	V	X	-
SpecialProperty	V	V	-
sex	V	X	-
weight	V	X	-
blood type	X	X	-
current location	X	X	-
number of siblings	V	X	-
happiness score	V	X	-
household income	V	X	-
conversations per day	V	X	-
sugar levels	V	X	-
sport activity	V	X	-
pcr date	X	X	-
PCR_01	V	X	Standard
PCR_02	V	X	Standard
PCR_03	V	X	Min-Max
PCR_04	V	X	Min-Max
PCR_05	V	X	Standard
PCR_06	V	X	Standard
PCR_07	V	X	Standard
PCR_08	V	X	Standard
PCR_09	V	X	Min-Max
PCR_10	V	X	Min-Max

Note: do not include the target variables "`spread`" and "`risk`" in the table.

Now, let's create an automatic data preparation pipeline for preparing incoming data for prediction. This pipeline should be based **only on the training set.** For example, if you normalized a feature using the standard scaler, you should calculate the mean and std from the training data, and apply this normalization to the new data.

Task H: Write a module[2] called `prepare.py` containing a function with the following signature:

```
def prepare_data(training_data, new_data)
```

The `new_data` parameter is the `DataFrame` to be prepared and `training_data` is the training set `DataFrame` used during data exploration. Your function should perform as described in **(Q23)**. The output is a copy of `new_data` (the original parameter should remain unchanged), after it has been preprocessed according to the provided `training_data`.

You are required to submit `prepare.py`.

Apply the function to both the train and test sets like so:

```
# Prepare training set according to itself
train_df_prepared = prepare_data(train_df, train_df)

# Prepare test set according to the raw training set
test_df_prepared = prepare_data(train_df, test_df)
```

Save your two preprocessed `DataFrame`s as CSV files and keep them for the next assignment. Do not submit any CSV files!

Important: Return to the instructions at the beginning of the document and make sure that you submit all the required files!

[2] If you are using jupyter notebook or Colab and have issues importing external modules, you can simply write the function in your notebook and copy it later to the `prepare.py` file using your preferred text editor. Do not forget to copy the relevant `import` statements that are required for your function to run.