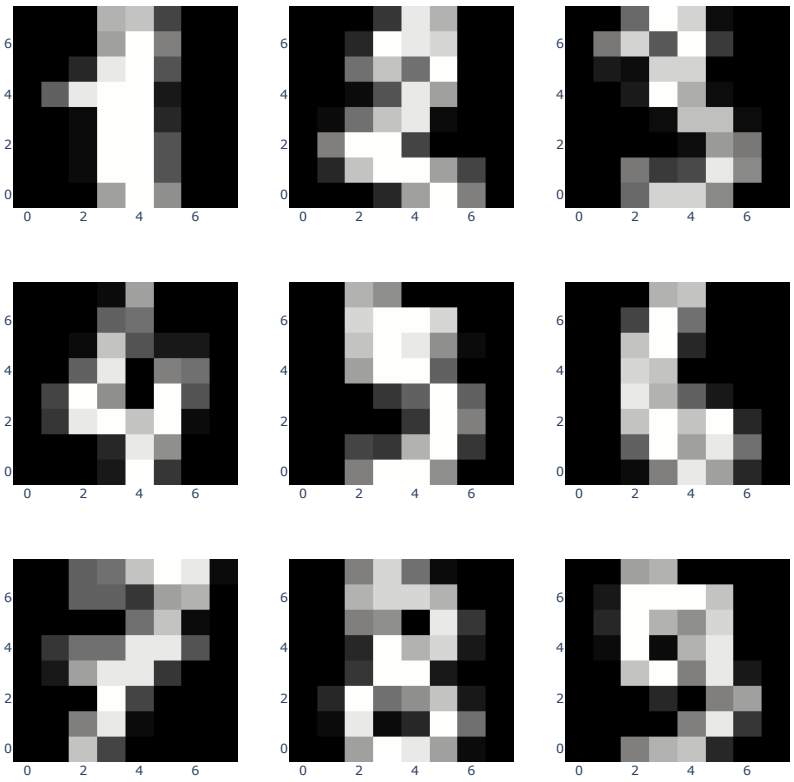
Machine Learning Module

Homework Report

Samuel Diebolt



ESPCI PARIS | PSL

Teacher: Olivier Rivoire

1 Course questions

1.1 Maximum likelihood estimation

Alice throws a coin 100 times and obtains 55 times a tail. Estimate by maximum likelihood the probability that the coin gives a tails. What confidence do we have in this result? Should Alice consider the coin to be unfair?

We're modelling the outcome of a coin flip by a Bernoulli distribution

$$p_{\text{model}}(x;\theta) = \theta^x(1-\theta)^{1-x}, \tag{1}$$

where the parameter θ represents the probability of getting a tails. Thus, the maximum likelihood estimator for θ is

$$\begin{aligned}\theta_{\text{ML}} &= \arg\max_\theta \mathrm{E}_{x\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(x;\theta)] \\ &= \arg\max_\theta \left(\hat{\theta}\log\theta + (1-\hat{\theta})\log(1-\theta)\right) \\ &= \hat{\theta},\end{aligned} \tag{2}$$

where $\hat{\theta}$ is the sample estimate of the probability that the coin gives a tails and $\hat{p}_{\text{data}}$ is the empirical distribution defined by the observed sample.

Given that after 100 throws, Alice obtained 55 tails, then

$$\theta_{\text{ML}} = \hat{\theta} = 0.55. \tag{3}$$

Knowing that

$$\begin{aligned}\mathrm{SE}(\theta_{\text{ML}}) &= \sqrt{\frac{\hat{\theta}(1-\hat{\theta})}{n}} \\ &\approx 0.05\end{aligned} \tag{4}$$

and considering the normal assumption ($100 \times 0.55 > 30$), the 95% confidence interval is given by

$$\theta_{\text{ML}} \in \theta_{\text{ML}} \pm 1.96 \times \mathrm{SE}(\theta_{\text{ML}}) \approx [0.45, 0.65]. \tag{5}$$

Therefore, as the 95% confidence interval contains 0.5, Alice cannot reject the null hypothesis that her coin is fair (at the 5% confidence level).

1.2 Bayesian estimation

Bob is tested for a disease. The test, which is either positive or negative, is only 90% reliable. Given that 1% of people of Bob's age and background have the disease, what is the probability that Bob has the disease?

Since we know nothing about the test results, we will assume that the test came back positive (without this assumption, the only valid conclusion would be that Bob has a 1% probability of having the disease).

Considering the events

- **ill**: "Bob has the disease",
- **+**: "Bob has tested positive",

and given that

$$\begin{cases} P(+|\text{ill}) = 0.90, \\ P(\text{ill}) = 0.01, \end{cases} \tag{6}$$

then using Bayes' theorem,

$$\begin{aligned} P(\text{ill}|+) &= \frac{P(\text{ill}, +)}{P(+)} \\ &= \frac{P(+|\text{ill})P(\text{ill})}{P(+|\text{ill})P(\text{ill}) + P(+|\overline{\text{ill}})P(\overline{\text{ill}})} \\ &= \frac{0.90 \times 0.01}{0.90 \times 0.01 + 0.10 \times 0.99} \\ &\approx 0.08. \end{aligned} \tag{7}$$

Thus, given that Bob has tested positive, he has a probability of about 8% to have the disease. This low probability reflects the lack of information we have (in fact, in a medical environment, the test result would be called *reactive* instead of *positive* and further testing would be done). Obvisouly, other factors (contact with contagious people, etc.), if taken into account in a refined model, could help Bob know if the result was a false or a true positive.

For example, a second positive test, given that the prior $P(\text{ill})$ has been updated to 8%, would give a probability of about 44% that Bob has the disease. A third positive test would increase this probability to 88%.

> *By redesigning the test, you can either reduce from 10% to 5% the false positive rate (less negative results when the patient is positive) or reduce from 10% to 5% the false negative rate (less positive results when the patient is negative): what is preferable?*

In the field of disease testing, false negatives are usually considered worse than false positives. Taking COVID-19 as an example: a false positive would lead to a heavy mental burden for the patient, but further testing would eventually show that they aren't ill, while a false negative would mean that the patient would be missing out on crucial treatments and runs a risk of spreading the disease. Therefore, decreasing the false negative rate is preferable for this test.

1.3 Information theory

The binary erasure channel is a discrete memoryless channel where each input $x_i \in \{0, 1\}$ is either transmitted reliably, with probability $1 - \epsilon$, or replaced by an error symbol $\star$, with probability ϵ. What is the capacity of this channel?

For a binary erasure channel with a probability ϵ of transmitting an error,

$$\begin{aligned} I(X; Y) &= H(X) - H(X|Y) \\ &= H(X) - P(y = 0)H(X|y = 0) - P(y = 1)H(X|y = 1) - P(y = \star)H(X|y = \star). \end{aligned} \tag{8}$$

Given $y \neq \star$, X is entirely determined by Y, thus

$$H(X|y = 0) = H(X|y = 1) = 0. \tag{9}$$

Also,

$$P(y = \star) = \epsilon \tag{10}$$

by definition of a binary erasure channel, and

$$H(X|y = \star) = H(X) \tag{11}$$

by symmetry (knowing that $y = \star$ doesn't give any information about X, so $P(X|y = \star) = P(X)$). Therefore,

$$I(X; Y) = (1 - \epsilon)H(X). \tag{12}$$

Finally, the capacity of a binary erasure channel is

$$\begin{aligned} C &= \sup_{p_X} I(X; Y) \\ &= (1 - \epsilon) \sup_{p_X} H(X) \\ &= 1 - \epsilon. \end{aligned} \tag{13}$$

The last equation holds if X comes from a Bernoulli distribution with parameter $\frac{1}{2}$, as the symmetry would suggest.

More generally, a memoryless erasure channel takes inputs from an alphabet of q symbols $\{1, 2, \dots, q\}$: any of these symbols is transmitted reliably with probability $1 - \epsilon$ and replaced by an error symbol $\star$ with probability ϵ. What is the capacity of this channel?

A memoryless erasure channel taking inputs from an alphabet of q symbols would have the same capacity as a binary erasure channel. This result is analogous to the previous proof, where X now comes from a categorical distribution (generalized Bernoulli) with parameter $\frac{1}{q}$ and the entropy H_q is now computed using a logarithm to base q.

1.4 Maximum entropy method

Consider N binary sequences of length p: $\sigma_{i,j} = \pm 1$ with $i = 1, \dots, N$ and $j = 1, \dots, p$. We use the maximum entropy method to estimate $P(\sigma_1, \dots, \sigma_p)$. Show that if we choose to constrain for each j the average of σ_j to the empirical mean $\mu_j = \sum_i \frac{\sigma_{i,j}}{N}$, the maximum entropy principle leads to a distribution of the form.

$$P(\sigma_1, \dots, \sigma_p) = \frac{1}{Z} \exp\left(\sum_{j=1}^{p} \beta_j \sigma_j \right). \tag{14}$$

Let $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_p)$. The first constraint on the unknown probability distribution is that its sum must normalize to 1,

$$\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) = 1. \tag{15}$$

Additionally, the first moment of each variable σ_j is supposed to match the value of the corresponding sample mean over the N measurements,

$$\langle \sigma_j \rangle = \sum_{\sigma \in \{-1,1\}} P_{\sigma_j}(\sigma)\sigma = \frac{1}{N} \sum_{i=1}^{N} \sigma_{ij} = \overline{\sigma_j}, \tag{16}$$

where P_{σ_j} is the marginal probability distribution of σ_j. Further deriving the first moment,

$$\begin{aligned} \langle \sigma_j \rangle &= \sum_{\sigma \in \{-1,1\}} P_{\sigma_j}(\sigma)\sigma \\ &= \sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma})\sigma_j. \end{aligned} \tag{17}$$

Finally, the probability distribution should maximize the information entropy:

$$S = -\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) \log P(\boldsymbol{\sigma}). \tag{18}$$

Introducing the frequencies $f_j \equiv \overline{\sigma_j}$, the corresponding Lagrangian $\mathcal{L} = \mathcal{L}(P(\boldsymbol{\sigma}); \alpha, \{\beta_j\})$ has the functional form

$$\mathcal{L} = S + \alpha \left(\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) - 1 \right) + \sum_{j=1}^{p} \beta_j \left(\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma})\sigma_j - f_j \right) \tag{19}$$

and its stationary point is found at

$$\begin{aligned} \frac{\partial \mathcal{L}}{P(\boldsymbol{\sigma})} &= -\log P(\boldsymbol{\sigma}) - 1 + \alpha + \sum_{j=1}^{p} \beta_j \sigma_j = 0 \\ \iff P(\boldsymbol{\sigma}) &= \frac{1}{Z} \exp\left(\sum_{j=1}^{p} \beta_j \sigma_j \right), \end{aligned} \tag{20}$$

where $Z \equiv e^{1-\alpha}$ is the partition function introduced as normalization constant.

What are the values of β_j and Z?

First, let us demonstrate the following identity:

$$\sum_{\boldsymbol{\sigma}} \exp\left(\sum_{j=1}^{p} \beta_j \sigma_j\right) = \prod_{j=1}^{p} \sum_{\sigma_j \in \{-1,1\}} \exp\left(\beta_j \sigma_j\right). \tag{21}$$

This comes easily by rewriting the left hand sum,

$$\begin{aligned}
\sum_{\boldsymbol{\sigma}} \exp\left(\sum_{j=1}^{p} \beta_j \sigma_j\right) &= \sum_{\boldsymbol{\sigma}} \prod_{j=1}^{p} \exp\left(\beta_j \sigma_j\right) \\
&= \sum_{\sigma_1 \in \{-1,1\}} \cdots \sum_{\sigma_p \in \{-1,1\}} \prod_{j=1}^{p} \exp\left(\beta_j \sigma_j\right) \\
&= \left(\sum_{\sigma_1 \in \{-1,1\}} \exp(\beta_1 \sigma_1)\right) \cdots \left(\sum_{\sigma_p \in \{-1,1\}} \exp\left(\beta_p \sigma_p\right)\right) \\
&= \prod_{j=1}^{p} \sum_{\sigma_j \in \{-1,1\}} \exp\left(\beta_j \sigma_j\right).
\end{aligned} \tag{22}$$

Now, recall that we required the probability distribution to follow the constraints,

$$\begin{cases}
\displaystyle\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) = 1, & \text{(23a)} \\
\displaystyle\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) \sigma_j = f_j, \qquad \text{for j} = 1, \ldots, \text{p.} & \text{(23b)}
\end{cases}$$

Using identity (21), equation (23a) leads to

$$\begin{aligned}
&\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) = 1 \\
\iff &\sum_{\boldsymbol{\sigma}} \frac{1}{Z} \exp\left(\sum_{j=1}^{p} \beta_j \sigma_j\right) = 1 \\
\iff &\frac{1}{Z} \prod_{j=1}^{p} \sum_{\sigma_j \in \{-1,1\}} \exp\left(\beta_j \sigma_j\right) = 1 \\
\iff &Z = \prod_{j=1}^{p} 2 \cosh \beta_j.
\end{aligned} \tag{24}$$

For $k = 1, \dots, p$, using once again identity (21), equation (23b) leads to

$$\begin{aligned}
&\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma})\sigma_k = f_k \\
\iff &\sum_{\boldsymbol{\sigma}} \left(\prod_{j=1}^{p} \exp\left(\beta_j \sigma_j\right) \right) \sigma_k = f_k Z \\
\iff &\left(\prod_{j=1; j\neq k}^{p} \sum_{\sigma_j \in \{-1,1\}} \exp\left(\beta_j \sigma_j\right) \right) \times \sum_{\sigma_k \in \{-1,1\}} \exp(\beta_k \sigma_k)\sigma_k = f_k Z \\
\iff &\frac{Z}{2\cosh(\beta_k)} \times 2\sinh(\beta_k) = f_k Z \\
\iff &\beta_k = \tanh^{-1} f_k.
\end{aligned} \tag{25}$$

Therefore, introducing this last result into equation (24) leads to

$$Z = \prod_{j=1}^{p} \frac{2}{\sqrt{1 - f_j^2}}. \tag{26}$$

Finally, these results yield the probability distribution assigned to the p binary sequences with $\sigma_j \in \{-1, 1\}$,

$$P(\boldsymbol{\sigma}) = \frac{1}{Z} \exp\left(\sum_{j=1}^{p} \beta_j \sigma_j \right), \tag{27}$$

where

$$\begin{cases} \beta_j = \tanh^{-1} f_j, \\ Z = \displaystyle\prod_{j=1}^{p} \frac{2}{\sqrt{1 - f_j^2}}. \end{cases} \tag{28}$$

What if we take $\sigma_{i,j} \in \{0, 1\}$ instead of $\sigma_{i,j} \in \{-1, 1\}$?

If $\sigma_{i,j} \in \{0, 1\}$, only the values of β_j and Z changes. Equation (23a) now leads to

$$\begin{aligned}
&\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) = 1 \\
\iff &\sum_{\boldsymbol{\sigma}} \frac{1}{Z} \exp\left(\sum_{j=1}^{p} \beta_j \sigma_j \right) = 1 \\
\iff &\frac{1}{Z} \prod_{j=1}^{p} \sum_{\sigma_j \in \{0,1\}} \exp\left(\beta_j \sigma_j\right) = 1 \\
\iff &Z = \prod_{j=1}^{p} (1 + \exp \beta_j).
\end{aligned} \tag{29}$$

For $k = 1, \dots, p$, equation (23b) now leads to

$$\begin{aligned}
&\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma})\sigma_k = f_k \\
\iff &\sum_{\boldsymbol{\sigma}} \left(\prod_{j=1}^{p} \exp\left(\beta_j \sigma_j\right) \right) \sigma_k = f_k Z \\
\iff &\left(\prod_{j=1; j\neq k}^{p} \sum_{\sigma_j \in \{-1,1\}} \exp\left(\beta_j \sigma_j\right) \right) \times \sum_{\sigma_k \in \{0,1\}} \exp(\beta_k \sigma_k)\sigma_k = f_k Z \\
\iff &\frac{Z}{1 + \exp \beta_k} \times \exp \beta_k = f_k Z \\
\iff &\beta_k = \log\left(\frac{f_k}{1 - f_k}\right).
\end{aligned} \tag{30}$$

Therefore, introducing this last result into equation (29) leads to

$$Z = \prod_{j=1}^{p} \frac{1}{1 - f_j}. \tag{31}$$

Finally, for binary sequences with $\sigma_j \in \{0, 1\}$, the probability distribution in equation (27) now has parameters

$$\begin{cases}
\beta_j = \log\left(\dfrac{f_j}{1 - f_j}\right), \\
Z = \displaystyle\prod_{j=1}^{p} \frac{1}{1 - f_j}.
\end{cases} \tag{32}$$

It is reassuring to see that this probability distribution can be written as

$$P(\boldsymbol{\sigma}) = \prod_{j=1}^{p} f_j^{\sigma_j} (1 - f_j)^{1-\sigma_j}, \tag{33}$$

which is a product of Bernoulli probability distributions with parameter f_j. These are in fact the marginal probability distributions of the σ_j variables: this last result could have been derived directly from the fact that the σ_j are mutually independent.

> *What if we also constrain the pair frequencies f_{jk} at which $\sigma_j = 1$ and $\sigma_k = 1$ co-occur for every pair (j, k)?*

The constraints are now

$$\begin{cases}
\displaystyle\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) = 1, & \text{(34a)} \\
\displaystyle\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma})\sigma_j = f_j, \qquad \text{for j = 1, …, p,} & \text{(34b)} \\
\displaystyle\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma})\sigma_j\sigma_k = f_{jk}, \qquad \text{for j,k = 1, …, p,} & \text{(34c)}
\end{cases}$$

where we introduced the pairwise frequencies $f_{jk} = \overline{\sigma_j \sigma_k}$,

$$f_{jk} \equiv \overline{\sigma_j \sigma_k} = \frac{1}{N} \sum_{i=1}^{N} \sigma_{i,j} \sigma_{i,k}. \tag{35}$$

The corresponding Lagrangian $\mathcal{L} = \mathcal{L}(P(\boldsymbol{\sigma}); \alpha, \{\beta_j\}, \{\gamma_{jk}\})$ has the functional form

$$\mathcal{L} = S + \alpha \left(\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) - 1 \right) + \sum_{j=1}^{p} \beta_j \left(\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) \sigma_j - f_j \right) + \sum_{j,k=1}^{p} \gamma_{jk} \left(\sum_{\boldsymbol{\sigma}} P(\boldsymbol{\sigma}) \sigma_j \sigma_k - f_{jk} \right) \tag{36}$$

and its stationary point is found at

$$\begin{aligned} \frac{\partial \mathcal{L}}{P(\boldsymbol{\sigma})} &= -\log P(\boldsymbol{\sigma}) - 1 + \alpha + \sum_{j=1}^{p} \beta_j \sigma_j + \sum_{j,k=1}^{p} \gamma_{jk} \sigma_j \sigma_k = 0 \\ \iff P(\boldsymbol{\sigma}) &= \frac{1}{Z} \exp\left(\sum_{j=1}^{p} \beta_j \sigma_j + \sum_{j,k=1}^{p} \gamma_{jk} \sigma_j \sigma_k \right), \end{aligned} \tag{37}$$

where $Z \equiv \exp(1 - \alpha)$ is the partition function.

2 Lasso regression and model selection: what makes a good wine?

This section will study the qualities of *vinho verde*, a wine from the Minho (northwest) region of Portugal. The provided datasets (Cortez *et al.*, 2009) are available to download on the author's website and contain twelve features of a sample of red and white *vinho verde* wines, ranging from objective test results (e.g. pH values) to sensory data (evaluations made by wine experts).

The goal of this section is to preprocess and use these datasets to

- find a predictive model of red wine quality using lasso regression;
- implement a classifier that will be able to recognize red and white wines from their physicochemical properties.

Listing 1 shows the Python libraries and plot rendering settings used to run all code blocks in this report.

```
# Python libraries.
from IPython.display import display
import numpy as np
import pandas as pd
import plotly.figure_factory as ff
import plotly.graph_objects as go
import plotly.io as pio
from plotly.subplots import make_subplots
from scipy import stats
from scipy.cluster.hierarchy import linkage
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.linear_model import lasso_path, Lasso, LassoCV
from sklearn.manifold import TSNE
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
import statsmodels.api as sm

# Set random generator seed.
np.random.seed(42)

# Set up plotly renderer.
png_renderer = pio.renderers["svg"]
png_renderer.width = 900
png_renderer.height = 700

pio.renderers.default = "svg"
```

Listing 1: Python libraries import.

2.1 Preprocessing

Figure 1 shows the different features available in the `winequality-red` dataset, displayed using Listing 2

```
# Read red wine quality dataset and display table head.
red_wine = pd.read_csv("data/winequality/winequality-red.csv",
                       sep=";")
red_wine.head()
```

Listing 2: Print red wine dataset header.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Figure 1: Red wine quality dataset header.

Figure 2 shows histograms of these features. Some features contain outliers that could introduce bias in the following analyses: a preprocessing step, performed in Listing 3, will remove values outside the ±3 standard deviations range.

```
# Plot histograms for each variable.
fig = make_subplots(rows=4, cols=3)

for i, col in enumerate(red_wine.columns, 1):
  fig.add_trace(
    go.Histogram(
      x=red_wine[col],
      name=col
    ),
    row=int(np.ceil(i / 3)),
    col=i % 3 if i % 3 != 0 else 3
  )

fig.update_layout(
  title_text="Histograms Of Red Wine Features - Before Outlier
     Removal",
  height=800
)
fig.show()
```

Listing 3: Preprocessing step.

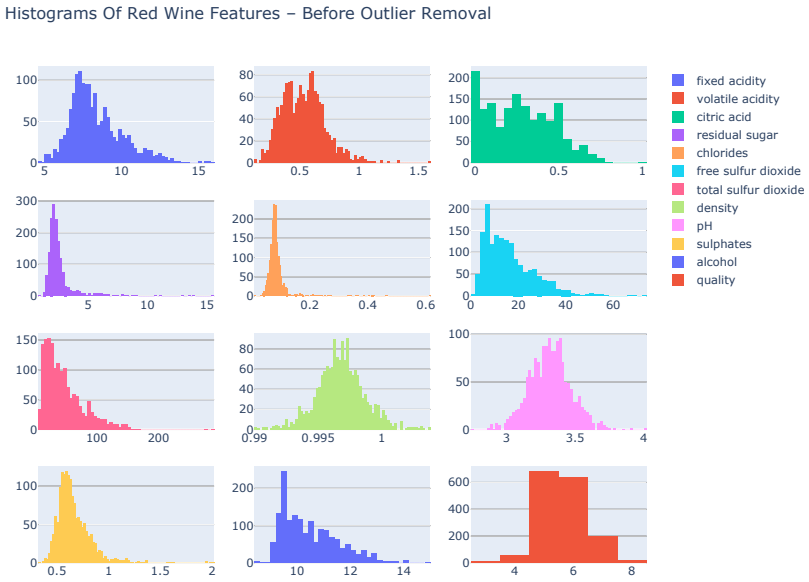


Figure 2: Red wine features histograms before outlier removal.

Figure 3 shows the histograms after outlier removal.

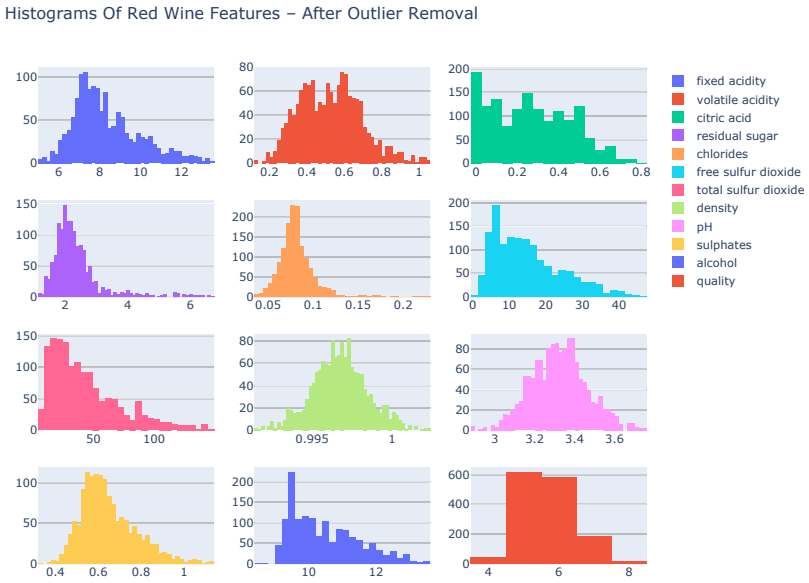


Figure 3: Red wine features histograms after outlier removal.

2.2 Principal component analysis

PCA is performed in Listing 4 to visualize correlations between the different features in the `winequality-red` dataset. All features are scaled to zero mean and unit variance beforehand (otherwise PCA might determine that the direction of maximal variance corresponds to features varying more than others because of their scales).

Figure 4 shows two plots used to study correlations:

- a scree plot, showing eigenvalues ordered from largest to smallest. This plot can be used to determine the number of eigenvalues to keep for a later analysis;
- a variables factor map, showing the correlations between all twelve features and the top two principal components. Features that are grouped together are positively correlated, while features on opposite sides of the unit circle are negatively correlated.

```
# Cleaned dataset is scaled to zero mean and unit variance.
red_wine_norm = pd.DataFrame(
  StandardScaler().fit_transform(red_wine_clean),
  columns=red_wine_clean.columns
)

# Fit PCA model.
pca_model = PCA()
pca_result = pca_model.fit_transform(red_wine_norm)

# Get correlations between components and features.
coef = pd.DataFrame(pca_model.components_.T[:, 0:2],
                    columns=["pc1", "pc2"],
                    index = red_wine.columns)

fig = make_subplots(rows=1, cols=2,
                    subplot_titles=("Scree Plot", "Variables Factor
                       Map"))

# Scree plot.
fig.add_trace(
  go.Bar(
    x=["PC {}".format(i) for i in range(1, pca_model.components_.
       shape[1]+1)],
    y=pca_model.explained_variance_ratio_,
    name="Scree Plot"
  ),
  row=1,
  col=1
)

# Variables factor map.
# The angle in the unit circle is used to compute the point color.
   This way
# correlated features will appear with similar colors.
```

```
fig.add_trace(
  go.Scatter(
    x=coef.pc1,
    y=coef.pc2,
    text=red_wine.columns,
    mode="markers+text",
    textposition="top center",
    marker=dict(
      size=10,
      color=np.angle(coef.pc1 + 1j*coef.pc2, deg=True)
    ),
    name="Variables Factor Map"
  ),
  row=1,
  col=2
)

# Add unit circle.
fig.update_layout(
  shapes=[
    # unfilled circle
    dict(
      type="circle",
      xref="x2",
      yref="y2",
      x0=-1,
      y0=-1,
      x1=1,
      y1=1
    ),
  ]
)

# Update figure size, axes titles and remove legend.
fig.update_layout(
  height=600,
  width=1200,
  showlegend=False,
  xaxis_title="Component Number",
  yaxis_title="Explained Variance Ratio",
  xaxis2_title="PCA 1 ({:.2f}%)".format(pca_model.
      explained_variance_ratio_[0]*100),
  yaxis2_title="PCA 2 ({:.2f}%)".format(pca_model.
      explained_variance_ratio_[1]*100)
)

fig.show(width=1400)
```

Listing 4: PCA and correlation plots.

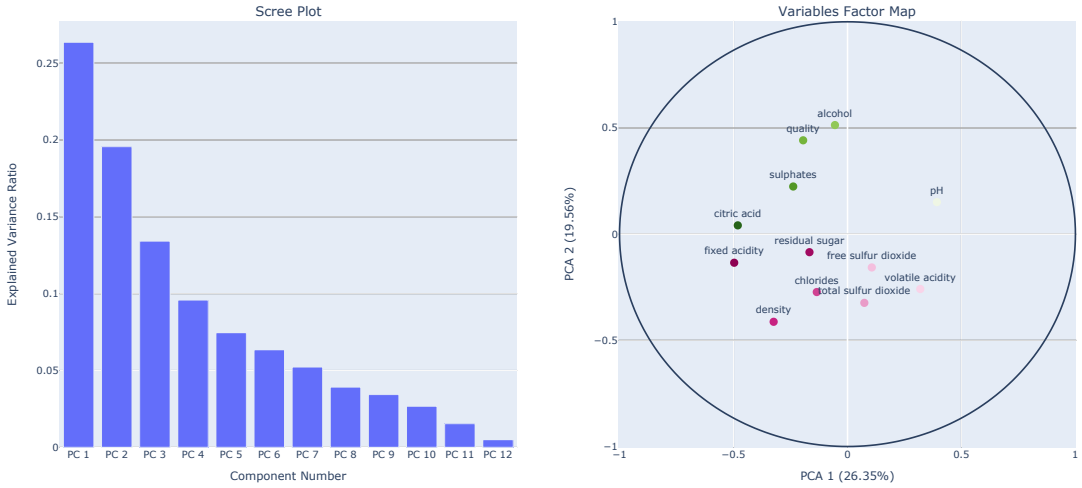


Figure 4: Scree plot and variables factor map from the principal component analysis of red wine features.

The scree plot shows that the top two principal components explain about 45% of the variance in the data. Therefore, using only the top two components to analyse the data is risky, as a lot of information is lost in the higher dimensions.

On the variables factor map, using the top two components, we observe:

- a strong positive correlation between the alcohol and quality features, with a slightly weaker correlation between these features and the sulphates.
- a strong negative correlation between the alcohol, quality and sulphates features and the volatile acidity, total sulfur dioxide and free sulfur dioxide features.

As previously stated, considering that the rest of the components still explain a significant part of the variance, the variables factor map should be interpreted with caution: in higher dimensions, some of these points could be farther apart than seen on a 2D visualization.

2.3 Multivariate linear regression

In Listing 5, we perform multivariate linear regression (ordinary least squares) of the `quality` score against the remaining 11 input features (with intercept).

```
# Standardize input features (zero mean, with or without unit
    variance).
X_mean_removed = StandardScaler(with_std=False).fit_transform(
    red_wine_clean.drop("quality", axis=1))
```

```
X_norm = StandardScaler().fit_transform(red_wine_clean.drop("quality", axis=1))

y = red_wine_clean.quality

# Perform multivariate linear regression.
est_zero_mean = sm.OLS(y, sm.add_constant(X_mean_removed)).fit()
est_norm = sm.OLS(y, sm.add_constant(X_norm)).fit()

# Plot regression coefficients as grouped bar plots.
# Significant coefficients are colored in red, non-significant in gray.
fig = make_subplots(
  rows=2, cols=1,
  shared_xaxes=True,
  vertical_spacing=0.1,
  subplot_titles=("Zero Mean", "Zero Mean, Unit Variance")
)

fig.add_trace(
  go.Bar(
    x=red_wine.columns,
    y=est_zero_mean.params[1:],
    name="Zero Mean",
    marker_color=["crimson" if p <= 0.05 else "lightslategray" for p in est_zero_mean.pvalues[1:]]
  ),
  row=1,
  col=1
)
fig.add_trace(
  go.Bar(
    x=red_wine.columns,
    y=est_norm.params[1:],
    name="Zero Mean, Unit Variance",
    marker_color=["crimson" if p <= 0.05 else "lightslategray" for p in est_norm.pvalues[1:]]
  ),
  row=2,
  col=1
)

fig.update_layout(
  title="Multivariate Linear Regression Coefficients, R^2 = {:.2f}, red: p < 0.05 ".format(est_norm.rsquared),
  showlegend=False,
  height=800
)
fig.show()
```

Listing 5: Multivariate linear regression of the quality score against all input features.

Figure 5 shows a comparison of the estimated model parameters when the input features have been standardized either by simply scaling to zero mean, or by by both scaling to zero mean and unit variance.

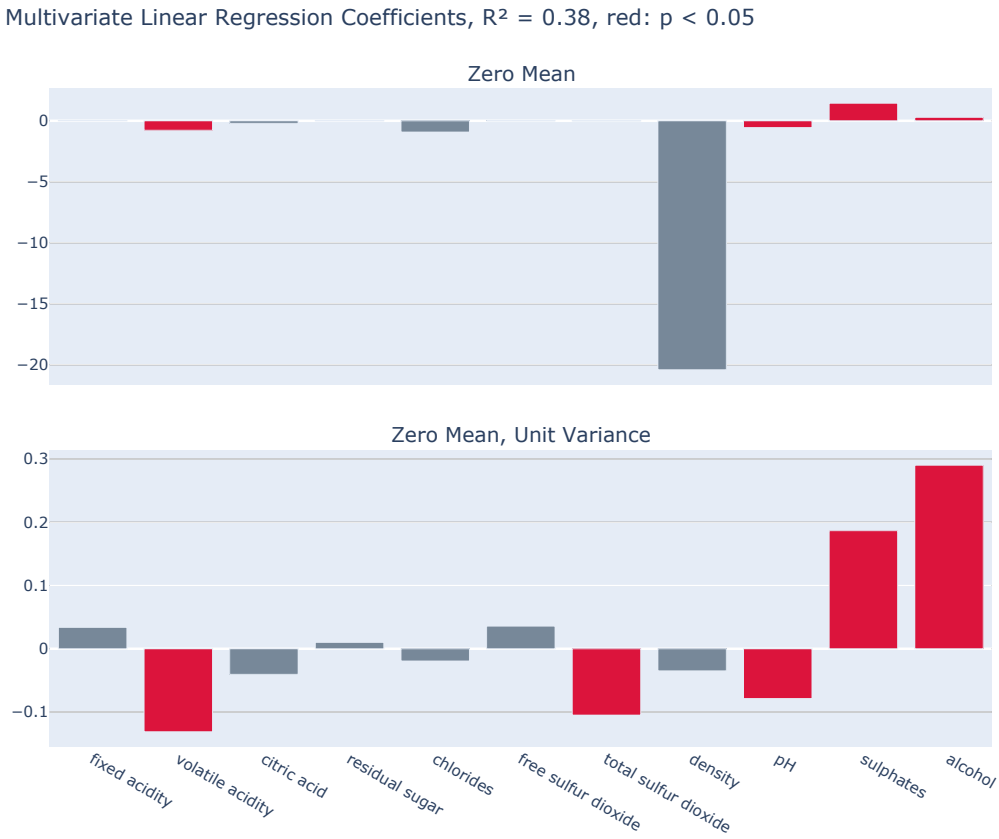


Figure 5: Bar plots comparing the estimated model parameters in two conditions. Significant parameters at the 5% confidence level are colored in red. (**Top**) input features have been standardized by scaling to zero mean. (**Down**) input features have been standardized by scaling to zero mean and unit variance.

The intercept term isn't displayed as it would hinder the visualization (it correponds to the mean of the `quality` score). As expected, the standard scaling of the dataset prior to regression (zero mean and unit variance) is essential to interpret the results: when the individual features aren't comparable due to their difference in scales, their respective regression coefficients aren't comparable either.

The coefficient of determination doesn't change between both settings, as it is already normalized (normalized covariance). Its value is relatively low: the current model including all input features and an intercept term explains only about 38% of the variance of the `quality` feature. Looking back at the histogram of the `quality` feature, we can see that it has very low variance, with most scores being either 5 or 6. Seeing that the difference in `quality` between the wines is almost binary, the multivariate linear model isn't well

suited. To obtain better predictions, further processing of the `quality` feature would be necessary (e.g. using a continuous scale), or a new model could be used (e.g. multivariate logistic regression, by binarizing the `quality` score).

Most regression coefficients aren't significant at the 5% level: the significant features are colored in red on the above box plots. From the remaining features, we observe that

- the `alcohol` and `sulphates` features show positive influence on the `quality` score;
- the `volatile acidity`, `total sulfur dioxide` and `pH` features show negative influence on the `quality` score.

2.4 Lasso regression and model selection

As the OLS model including all input features was inaccurate, lasso is suggested in this section as a way to select relevant features for the prediction of `quality`. However, lasso is likely to suffer the same fate as OLS, as it won't be able to explain more variance than the full OLS model.

In Listing 6, the lasso model is fitted along a given regularization path: $\alpha \in [10^{-3}, 1]$. To assess performance, the cleaned dataset is split into a training set of size 500 and a testing test of size 941. A scaler for zero mean and unit variance is fitted on the training set and the transformation applied on both training and testing sets. Finally, to find the best model, 10-fold cross-validation is used along the regularization path.

```
# Split dataset into testing and training set.
X_train, X_test, y_train, y_test = train_test_split(
  red_wine_clean.drop("quality", axis=1),
  red_wine_clean.quality,
  train_size=500,
  random_state=42
)

# Standardize data, fitting only on training set.
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

_, n = X_train.shape

# Regularization parameters path.
alphas = np.logspace(-3, 0, 100)

# Compute lasso path with coordinate descent.
alphas_lasso, coefs_lasso, _ = lasso_path(X_train, y_train, alphas=
    alphas, random_state=42)

# Compute lasso with iterative fitting along the regularization path.
# The best model is chosen by 10-fold cross-validation.
model = LassoCV(cv=10, alphas=alphas, random_state=42).fit(X_train,
    y_train)
```

```
25
26  # Compute training and testing score (R^2) along the regularization
        path.
27  training_score = []
28  testing_score = []
29  for a in alphas:
30    reg = Lasso(alpha=a, random_state=42).fit(X_train, y_train)
31    training_score.append(reg.score(X_train, y_train))
32    testing_score.append(reg.score(X_test, y_test))
33
34  # Plot lasso coefficients, MSE and performance along the
        regularization path.
35  fig = make_subplots(
36    rows=3,
37    cols=1,
38    vertical_spacing=0.1,
39    subplot_titles=("Lasso Path", "K-Fold Cross-Validation, K = 10", "
          Lasso Performance")
40  )
41
42  # Lasso coefficients.
43  for i in range(n):
44    fig.add_trace(
45      go.Scatter(
46        x=alphas_lasso,
47        y=coefs_lasso[i],
48        name=red_wine_clean.columns[i],
49        mode="lines",
50        legendgroup="lasso"
51      ),
52      row=1,
53      col=1
54    )
55
56  # Lasso MSE.
57  for i in range(10):
58    fig.add_trace(
59      go.Scatter(
60        x=model.alphas_,
61        y=model.mse_path_[:, i],
62        mode="lines",
63        line=dict(dash="dot"),
64        marker_color="grey",
65        showlegend=False,
66        name="MSE {}".format(i)
67      ),
68      row=2,
69      col=1
70    )
71  fig.add_trace(
72    go.Scatter(
73      x=model.alphas_,
```

```
    y=model.mse_path_.mean(axis=1),
    mode="lines",
    marker_color="black",
    legendgroup="cv",
    name="Average MSE"
  ),
  row=2,
  col=1
)

# Lasso performance.
fig.add_trace(
  go.Scatter(
    x=alphas,
    y=training_score,
    mode="lines",
    legendgroup="performance",
    name="Training"
  ),
  row=3,
  col=1
)
fig.add_trace(
  go.Scatter(
    x=alphas,
    y=testing_score,
    mode="lines",
    legendgroup="performance",
    name="Testing"
  ),
  row=3,
  col=1
)

# Add dashed line on each plot to indicate the regularization factor
   for the best model.
min_alpha = model.alphas_[np.argmin(model.mse_path_.mean(axis=1))]
fig.add_shape(
  go.layout.Shape(
    type="line",
    yref="y",
    xref="x",
    x0=min_alpha,
    y0=-0.2,
    x1=min_alpha,
    y1=0.43,
    line=dict(dash="dot")
  ),
  row=1,
  col=1
)
fig.add_shape(
```

```
  go.layout.Shape(
    type="line",
    yref="y",
    xref="x",
    x0=min_alpha,
    y0=-0.2,
    x1=min_alpha,
    y1=0.43,
    line=dict(dash="dot")
  ),
  row=1,
  col=1
)
fig.add_shape(
  go.layout.Shape(
    type="line",
    yref="y",
    xref="x",
    x0=min_alpha,
    y0=0.27,
    x1=min_alpha,
    y1=0.9,
    line=dict(dash="dot")
  ),
  row=2,
  col=1
)
fig.add_shape(
  go.layout.Shape(
    type="line",
    yref="y",
    xref="x",
    x0=min_alpha,
    y0=-0.05,
    x1=min_alpha,
    y1=0.4,
    line=dict(dash="dot")
  ),
  row=3,
  col=1
)

fig.update_layout(
  xaxis_type="log",
  xaxis2_type="log",
  xaxis3_type="log",
  xaxis3_title="Regularization Parameter",
  yaxis_title="Lasso coefficients",
  yaxis2_title="Validation Sets MSE",
  yaxis3_title="Model Performance (R^2)",
  height=800
)
```

```
177
178 fig.show()
```

Listing 6: Lasso regression of the quality score against all input features, along a given regularization path.

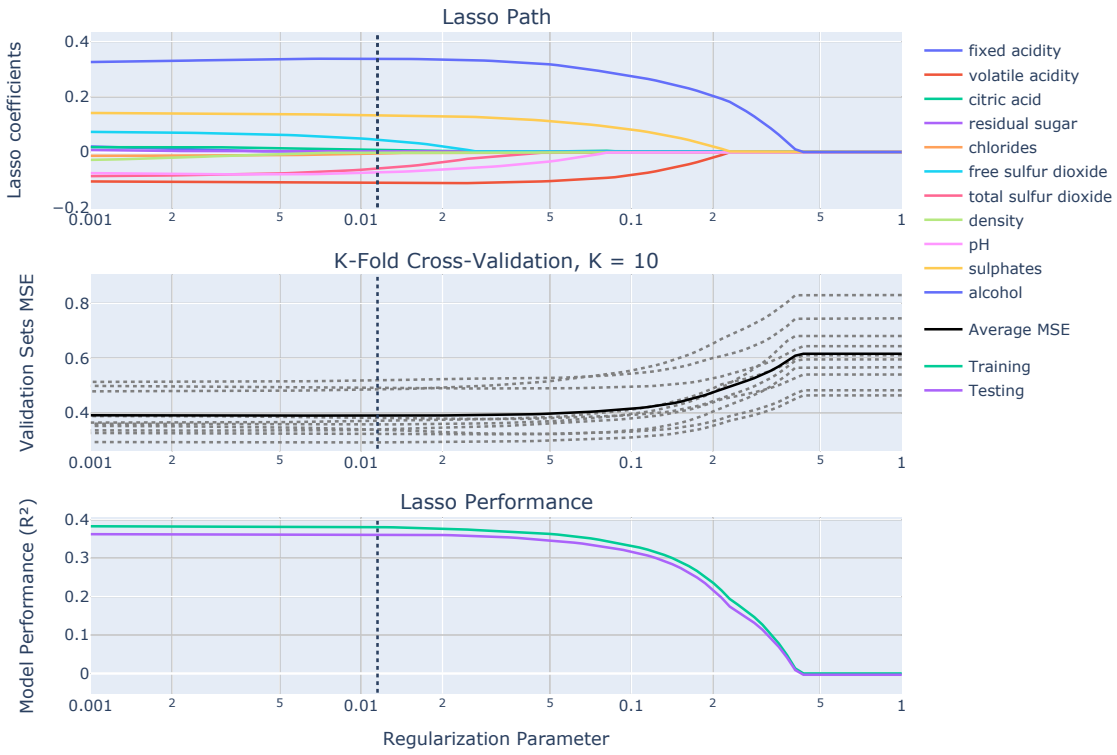


Figure 6: Lasso regression of the quality score against all input features, along a given regularization path. (**Top**) Evolution of the lasso coefficients with the regularization parameter. (**Middle**) Evolution of the mean squared error of each validation set from a 10-fold cross-validation. (**Bottom**) Evolution of lasso performance with the regularization parameter.

The dashed line across all plots of Figure 6 represents the best model obtained through 10-fold cross-validation. For this optimal model, the lasso regression mainly includes:

- the `alcohol` and `sulphates` features, contributing positively to the `quality`;
- the `volatile acidity` feature contributing negatively to the `quality`.

This result is very close to the significant features found in the OLS model. Unexpectedly, the performance of the optimal lasso model is similar to the performance of the previous OLS model. Moreover, the benefit of cross-validation on this task seems negligible, considering the flatness of the average MSE. We can once again conclude that a linear model isn't well suited to this task.

2.5 Next steps

Since the previous linear models were not able to accurately predict the `quality` of red wines, we are interested in knowing if a new model that takes into account the previous observations could do better. Since the `quality` is almost binary, as seen on the first histogram, we fix an arbitrary threshold in Listing 7 that will discriminate between *good* and *bad* wines. This threshold is fixed at a `quality` of 5.5, so that wines with a score of 5 or lower will be rated as *bad*. Figure 7 shows the counts for each quality.

```
# Deep copy used so that the original dataset isn't modified.
red_wine_bin = red_wine_clean.copy(deep=True)

# Binarize the quality feature into "good" and "bad" wines.
bins = (2, 5.5, 8)
group_names = ["bad", "good"]
red_wine_bin["quality"] = pd.cut(red_wine_bin.quality, bins=bins,
    labels=group_names)

# Bar plot of wine quality.
fig = go.Figure(go.Bar(
  x=["bad", "good"],
  y=red_wine_bin.quality.value_counts()
))

fig.update_layout(
  title="Red Wine Quality - Binarized",
  xaxis_title="Quality",
  yaxis_title="Count",
  height=800
)

fig.show(height=300)
```

Listing 7: Binarize the quality feature.

In Listing 8, the dataset is split into training and testing sets, with a more common 80%/20% ratio. The input features are standardized to get optimal results. A linear SVM model is used for classification, because of its ease of use.

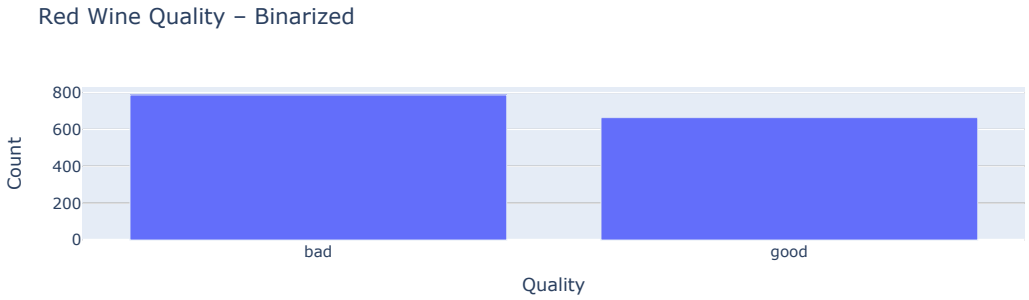


Figure 7: Red wine binarized quality counts.

```
# Split dataset into training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(
  red_wine_bin.drop("quality", axis=1),
  red_wine_bin.quality,
  test_size=0.2,
  random_state=42
)

# Standardize input features.
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

clf = LinearSVC(max_iter=2000)
clf.fit(X_train, y_train)

print("Classification report on the testing test:\n")
print(classification_report(y_test, clf.predict(X_test)))

#Classification report on the testing test:
#
#              precision    recall  f1-score   support
#
#         bad       0.69      0.69      0.69       131
#        good       0.74      0.74      0.74       160
#
#    accuracy                           0.72       291
#   macro avg       0.72      0.72      0.72       291
#weighted avg       0.72      0.72      0.72       291
```

Listing 8: Linear SVM classifier of red wine quality.

We obtain a mean precision and recall on the testing set of about 72%. This performance

could be slightly improved by using cross-validation and a non-linear SVM (kernel trick), but other methods (random forest, k-NN, etc.) should probably show better performances.

2.6 Classification

In this section, we will try to classify wine color from their physicochemical properties, using datasets `winequality-red` and `winequality-white`. In Listing 9, the white wine dataset is cleaned out of outliers using the same criterion as before (outliers outside the ±3 standard deviations range are removed) and merged with the clean red wine dataset. A `color` feature is added to the dataset to distinguish red from white wines. The `quality` feature is removed as it won't be used by the classifier. Figure 8 shows the features histograms for the merged datasets.

```
# Read white wine dataset and apply the same cleaning procedure as before.
white_wine = pd.read_csv("data/winequality/winequality-white.csv", sep=";")
white_wine_clean = white_wine[(np.abs(stats.zscore(white_wine)) < 3).all(axis=1)]

# Add color feature and merge datasets.
wine = pd.concat([red_wine_clean.assign(color="red"), white_wine_clean.assign(color="white")]).drop("quality", axis=1)

# Plot histograms for each variable.
fig = make_subplots(rows=4, cols=3)

for i, col in enumerate(wine.columns, 1):
  fig.add_trace(
    go.Histogram(
      x=wine[col],
      name=col
    ),
    row=int(np.ceil(i / 3)),
    col=i % 3 if i % 3 != 0 else 3
  )

fig.update_layout(
  title_text="Histograms Of Red and White Wine Features",
  height=800
)

fig.show()
```

Listing 9: Clean and merge white wine dataset.

Figure 8: Red and white wines features histograms after outlier removal.

With both datasets merged, we observe that some histograms are bimodal, which could indicate that these features will enable discrimination of red and white wines. Figure 9 shows a scatter plot of `chlorides` vs. `total sulfur dioxide`, two features that shows bimodal histograms, obtained using Listing 10.

```
# Scatter plot chlorides vs. total sulfur dioxide.
fig = go.Figure()

fig.add_trace(go.Scatter(
  x=wine.loc[wine.color == "white", "total sulfur dioxide"],
  y=wine.loc[wine.color == "white", "chlorides"],
  mode="markers",
  name="White Wines"
))
fig.add_trace(go.Scatter(
  x=wine.loc[wine.color == "red", "total sulfur dioxide"],
  y=wine.loc[wine.color == "red", "chlorides"],
  mode="markers",
  name="Red Wines"
))

fig.update_layout(
  title="Chlorides vs. Total Sulfur Dioxide",
  xaxis_title="Total Sulfur Dioxide",
  yaxis_title="Chlorides",
  height=800
```

```
)

fig.show()
```

Listing 10: Scatter plot of chlorides vs. total sulfur dioxide.

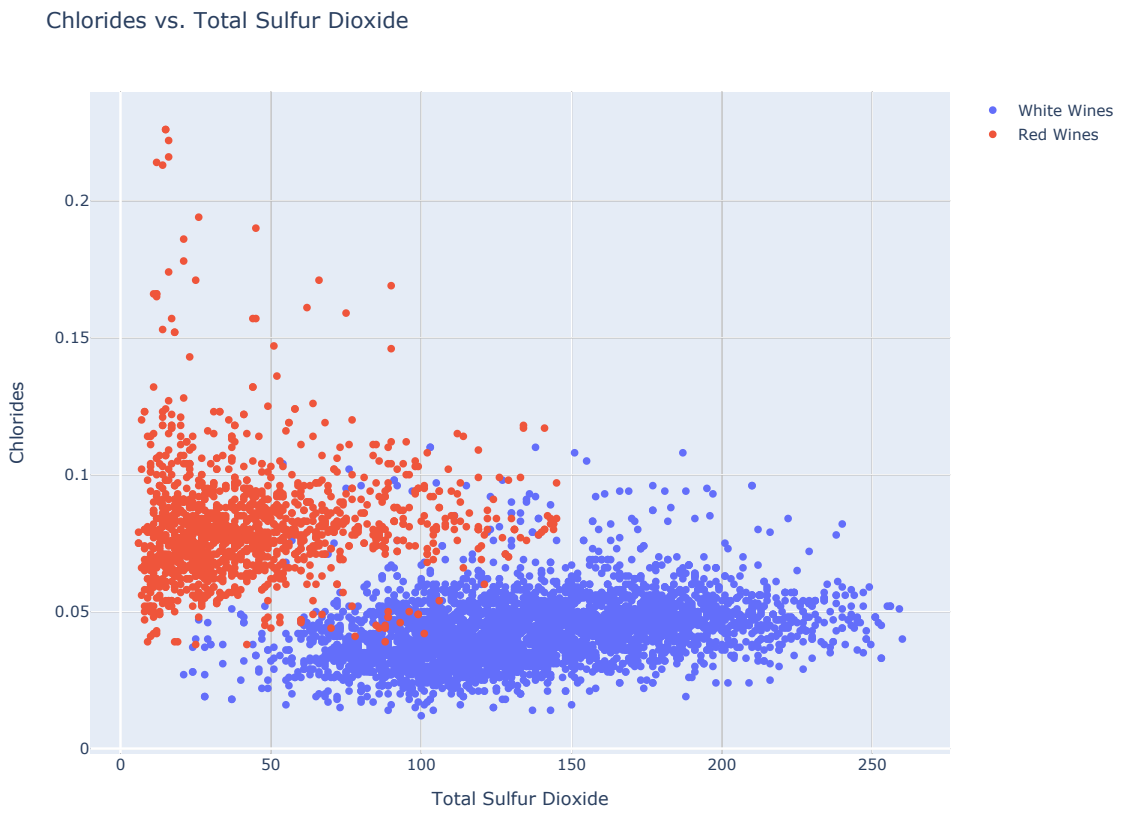


Figure 9: Scatter plot of chlorides vs. total sulfur dioxide.

Indeed, using only two features, wines can be almost perfectly visually classified according to their color. Given its ease of use compared to a PCA + k-mean clustering classifier, a linear SVM model is used once again in Listing 11. Considering the observations on the histograms, the dataset is split into training and testing sets using a 10%/90% ratio.

```
# Split dataset into training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(
  wine.drop("color", axis=1),
  wine.color,
  test_size=0.90,
  random_state=42
```

```
)

# Standardize input features.
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)

# Fit linear SVM and print accuracy.
clf = LinearSVC(max_iter=2000)
clf.fit(X_train, y_train)

print("Mean accuracy on training set: {:.2f}%".format(100*clf.score(
    X_train, y_train)))
print("Mean accuracy on testing set: {:.2f}%".format(100*clf.score(
    X_test, y_test)))

#Mean accuracy on training set: 100.00%
#Mean accuracy on testing set: 99.76%
```

Listing 11: Linear SVM classifier of wine color.

The linear SVM classifier is perfectly suited for this task: the error on the testing set is only about 0.2%. Finally, we can study which variables should be kept to keep this error below 1%. The `chlorides`, `total sulfur dioxide`, `density` and `residual sugar` features are selected based on their histograms and a new linear SVM classifier is fitted using this reduced dataset in Listing 12.

```
# Split dataset into training and testing sets.
wine_test = wine[["chlorides", "total sulfur dioxide", "density", "
    residual sugar", "color"]]

X_train, X_test, y_train, y_test = train_test_split(
    wine_test.drop("color", axis=1),
    wine_test.color,
    test_size=0.90,
    random_state=42
)

# Standardize input features.
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)

# Fit linear SVM and print accuracy.
clf = LinearSVC(max_iter=2000)
clf.fit(X_train, y_train)

```

```
print("Mean accuracy on training set: {:.2f}%".format(100*clf.score(
    X_train, y_train)))
print("Mean accuracy on testing set: {:.2f}%".format(100*clf.score(
    X_test, y_test)))

#Mean accuracy on training set: 99.33%
#Mean accuracy on testing set: 99.33%
```

Listing 12: Linear SVM classifier of wine color.

As we can see, keeping only these features, the test error is only about 0.7%: we were able to reduce the dataset to only four input features. Therefore, measuring only these four physicochemical properties should be sufficient to achieve a good color classification.

3 Clustering of handwritten digits

In this section, we consider the Handwritten Digits Data Set (E. Alpaydin *et al.*, 1998), available on the UC Irvine Machine Learning Repository, and in particular the `optdigits.tes` testing dataset.

The dataset was obtained by processing 32×32 normalized bitmaps of handwritten digits. These bitmaps were divided into nonoverlapping 4×4 blocks and the number of *on* pixels were counted in each block. This operation reduced each bitmap to an 8×8 input matrix where each element is an integer in the range $[0, 16]$. Each row of the `optidigits.tes` dataset contains an 8×8 matrix reshaped to a horizontal 1×64 vector, with an additional class attribute column specifying the digit.

The goal of this section is to cluster the data without taking into account the labels, thus studying an unsupervised learning method. The class attribute column containing the label will be used to assess the results.

In Listing 13, the data is read and the 64 input features are standardized to zero mean and unit variance. Figure 13 shows some digits examples from the dataset. As we can see, even with the dimension reduction applied by the authors, most digits are still identifiable.

```
# Read dataset and separate input features and labels.
dig = np.loadtxt("data/optdigits.tes", dtype='i', delimiter=',')
X = dig[:, :-1]
y = dig[:, -1]

n, m = X.shape

# Plot some examples from the dataset.
fig = make_subplots(rows=3, cols=3)

for i in range(1,10):
  fig.add_trace(
    go.Heatmap(
```

```
      z=np.flip(X[i, :].reshape((8, 8)), 0),
      showscale=False,
      colorscale="gray",
      name="Label: {}".format(y[i])
    ),
    row=int(np.ceil(i / 3)),
    col=i % 3 if i % 3 != 0 else 3
  )

fig.update_layout(
  title="Handwritten Digits Dataset, Examples",
  height=800,
  width=800
)

fig.show(height=900)

# Standardize input features.
X_norm = StandardScaler().fit_transform(X)
```

Listing 13: Read handwritten digits dataset and display example images.

Figure 10: Handwritten digits examples

3.1 K-means clustering

In this section, a K-means clustering method will be applied to cluster the data. First, we would like to find a good 2D visualization of the dataset. Figure 11 shows the projection of the data on the top two principal components obtained from a PCA performed in Listing 14.

```
# Perform PCA and plot the projected input features on top 2
    components.
pca_model = PCA()
pca_result = pca_model.fit_transform(X_norm)

fig = go.Figure()

for i in range(10):
  fig.add_trace(go.Scatter(
    x=pca_result[y==i, 0],
    y=pca_result[y==i, 1],
    mode="markers",
    marker_color=i,
    name="True Label: {}".format(i)
  ))

fig.update_layout(
  title="PCA Visualization Of Handwritten Digits",
  xaxis_title="PCA 1 ({:.2f}%)".format(pca_model.
      explained_variance_ratio_[0]*100),
  yaxis_title="PCA 2 ({:.2f}%)".format(pca_model.
      explained_variance_ratio_[1]*100),
  height=800
)

fig.show()
```

Listing 14: Perform PCA and project on top two components.

Unfortunately, individual clusters are hardly identifiable on this visualization as the top two components explain a very low variance ratio. The Handwritten Digits dataset being a non-trivial high-dimensional structure, this type of linear projections won't be able to perform well. Luckily, t-SNE, a popular visualization algorithm that is often very succesful at revealing clusters in data, can be used in this situation. t-SNE roughly works by trying to optimize for preserving the topology of the data. Figure 12, shows a t-SNE visualization of the dataset, projected on the top two components, obtained using Listing 15.

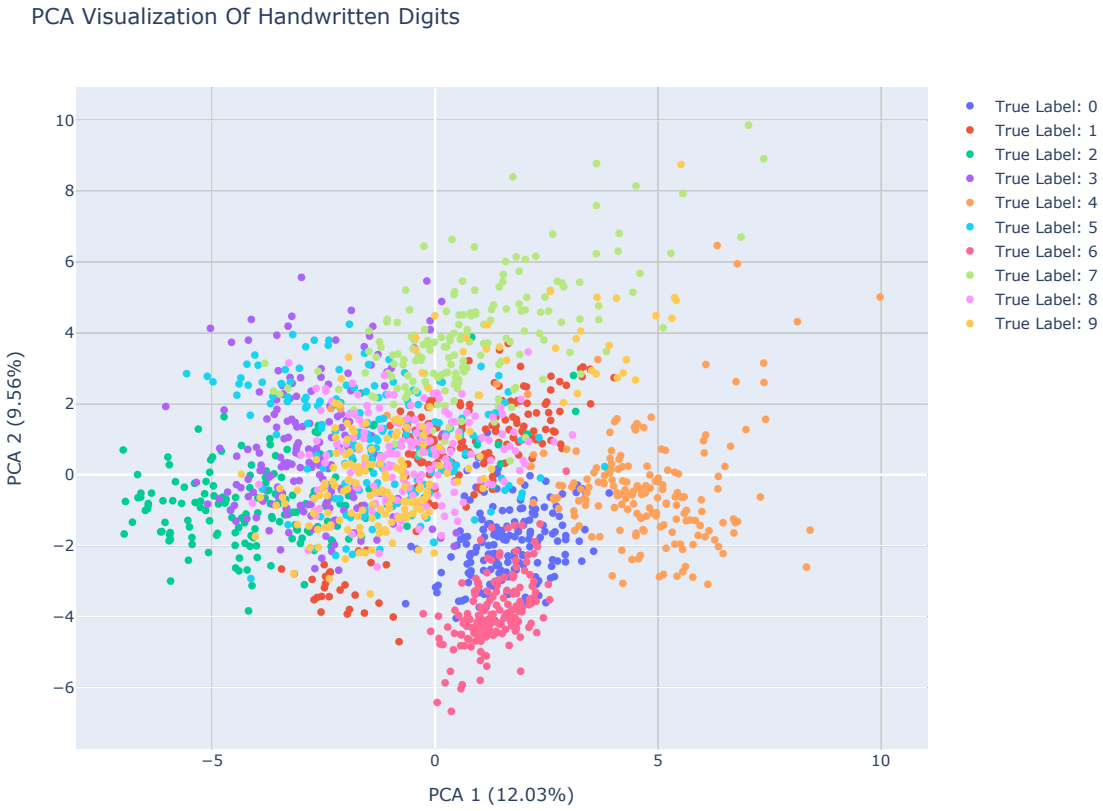


Figure 11: PCA visualization of handwritten digits.

```
# Fit t-SNE with PCA initialization.
tsne = TSNE(n_components=2, init="pca", random_state=42)
X_tsne = tsne.fit_transform(X_norm)

fig = go.Figure()

for i in range(10):
  fig.add_trace(go.Scatter(
    x=X_tsne[y==i, 0],
    y=X_tsne[y==i, 1],
    mode="markers",
    marker_color=i,
    name="True Label: {}".format(i)
  ))

fig.update_layout(
  title="t-SNE Visualization Of Handwritten Digits",
  height=800
)
```

```
20
21 fig.show()
```

Listing 15: Perform t-SNE and project on top two components.

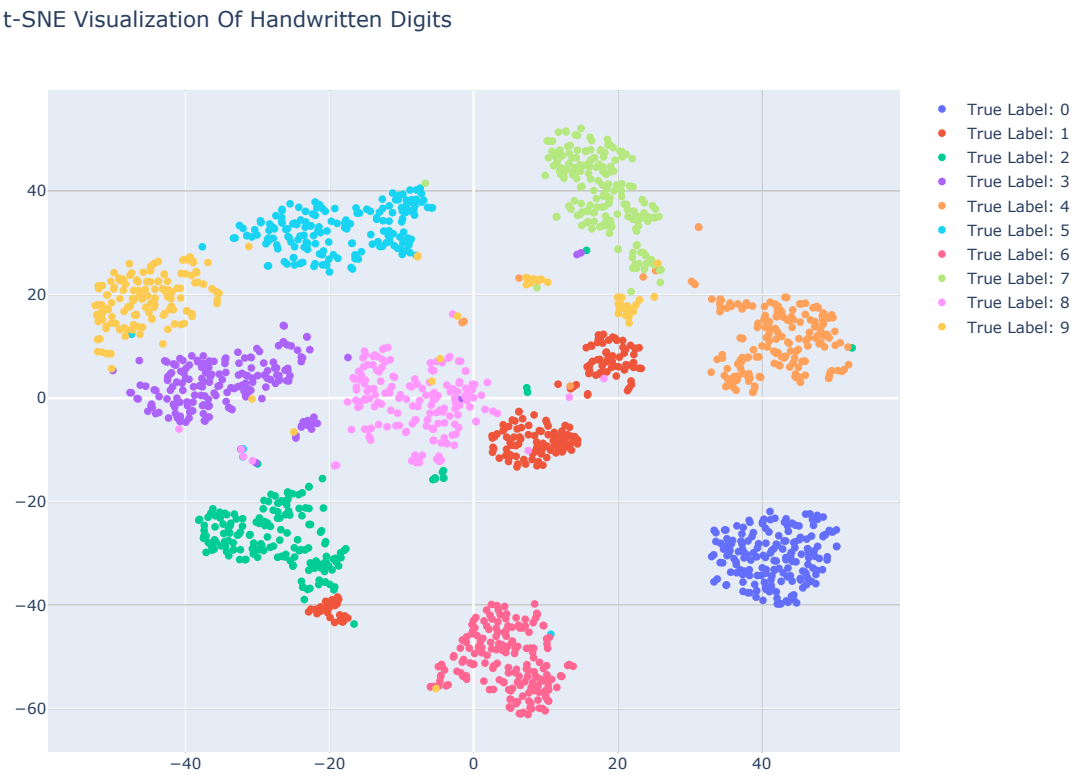


Figure 12: t-SNE visualization of handwritten digits.

As we can see, t-SNE does an impressive job at finding clusters compared to PCA. However, it is still prone to getting stuck in local minima, as is the case for some images of ones.

We can now perform K-means clustering and visualize its performance using t-SNE compared to Figure 12. The clustering is performed in Listing 16 using $K = 10$ clusters (for the 10 available digits) and the results of 10 different initial conditions are displayed on Figure 13. These initial conditions should not be chosen based on the t-SNE results, since neither distances nor density are preserved well (E. Schubert *et al.*, 2017). Therefore, initial cluster centroids will be selected at random from the observations.

To evaluate the clustering performance, two criterion will be compared:

- the intertia, corresponding to the sum of distances of samples to their closest cluster center;
- an arbitrary score, corresponding to the fraction of pairs of samples that are correctly partitioned.

```
# Store scores as defined by the exercise, and k-means inertias.
scores = []
inertias = []
for cond in range(10):
  # Perform k-means, using a fixed random state for reproducibility.
  kmeans = KMeans(n_clusters=10, init="random", n_init=1,
      random_state=cond).fit(X_norm)
  labels = kmeans.labels_

  # Compute clustering score.
  digit_pairs = np.zeros((n, n))
  for i in range(0, n):
    for j in range(i+1, n):
      digit_pairs[i,j] = (labels[i]==labels[j] and y[i]== y[j]) or (
          labels[i]!=labels[j] and y[i]!=y[j])
  scores.append(100*np.sum(digit_pairs)*2/(n*(n-1)))
  inertias.append(kmeans.inertia_)

print("Minimum score: {:.2f}%".format(np.min(scores)))
print("Maximum score: {:.2f}%".format(np.max(scores)))

# Plot normalized criteria.
fig = go.Figure()

fig.add_trace(go.Bar(
  x=list(range(1, 10)),
  y=(scores - np.mean(scores)) / np.std(scores),
  name="Clustering Score"
))

fig.add_trace(go.Bar(
  x=list(range(1, 10)),
  y=(inertias - np.mean(inertias)) / np.std(inertias),
  name="Clustering Inertia"
))

fig.update_layout(
  title="Normalized Clustering Score And Inertia",
  xaxis_title="Condition Number",
  height=800
)

fig.show()

#Minimum score: 88.47%
```

```
#Maximum score: 92.51%
```

Listing 16: Perform K-means clustering and compare performance between random initializations.

Figure 13: Comparison of K-means performance between 10 random initializations.

As we can see from Figure 13, the clustering score and inertia seem to be uncorrelated. The maximum score reached is about 93% and doesn't seem to vary much between conditions. Below is displayed the result of a K-means clustering with $K = 10$ and using the k-means++ algorithm with 10 repetitions for finding initial centroids (D. Arthur *et al.*, 2007).

```
# Perform k-means using 10 clusters, k-means++ and 10 repetitions.
kmeans = KMeans(n_clusters=10, random_state=42).fit(X_norm)

# Plot t-SNE visualization with infered labels.
fig = go.Figure()
```

```
6
7  for i in range(10):
8    fig.add_trace(go.Scatter(
9      x=X_tsne[kmeans.labels_==i, 0],
10     y=X_tsne[kmeans.labels_==i, 1],
11     mode="markers",
12     marker_color=i,
13     name="Infered Label: {}".format(i)
14   ))
15
16 fig.update_layout(
17   title="t-SNE Visualization Of Handwritten Digits, K-means
       clustering",
18   height=800
19 )
20
21 fig.show()
```

Listing 17: Visualize K-means results using t-SNE.

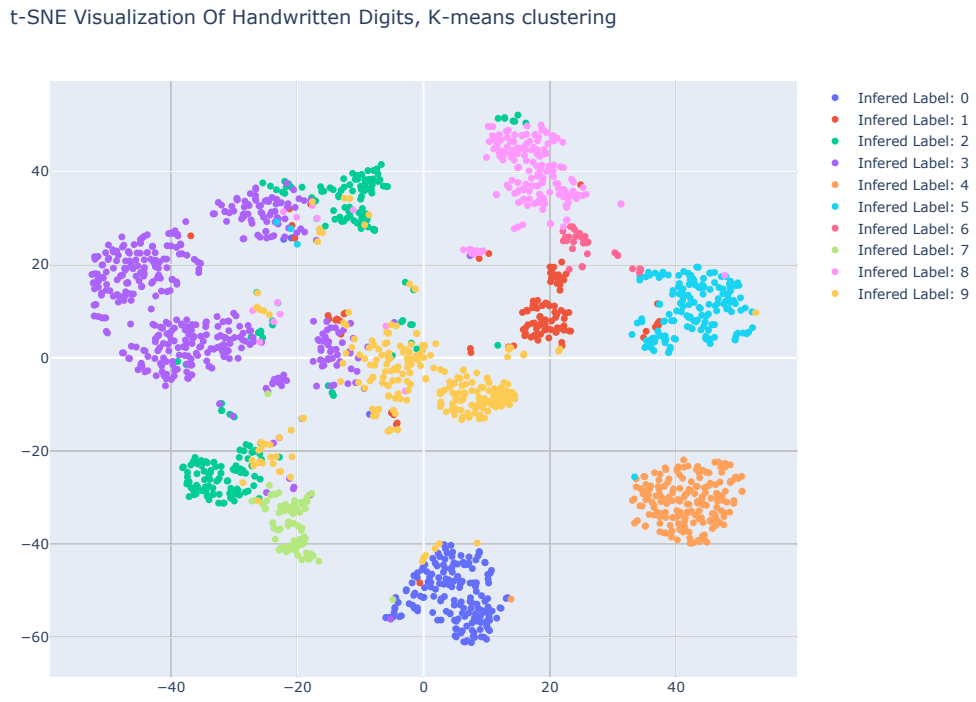


Figure 14: t-SNE visualization of handwritten digits, labels infered using K-means clustering.

By comparing with the first t-SNE visualization, the K-means clustering was able to correctly identify some of the clusters, but still fails on most of them. For example, the ones are mostly absorbed in the original cluster of eights, while the cluster of threes, fives and nines are merged into one cluster.

Considering that the correct number of clusters was known beforehand, we are now interested in knowing if this parameter can be infered from the data. The performance and inertia of K-means clusterings with K varying from 5 to 20 is compared using Listing 18 and the results are shown on Figure 15.

```
# Store custom score and k-means inertias.
scores = []
inertias = []
for n_clusters in range(5, 21):
  # Perform k-means, using a fixed random state for reproducibility.
  kmeans = KMeans(n_clusters=n_clusters, random_state=n_clusters).fit
      (X_norm)
  labels = kmeans.labels_

  # Compute clustering score.
  digit_pairs = np.zeros((n, n))
  for i in range(0, n):
    for j in range(i+1,n):
      digit_pairs[i,j] = (labels[i]==labels[j] and y[i]== y[j]) or (
          labels[i]!=labels[j] and y[i]!=y[j])
  scores.append(100*np.sum(digit_pairs)*2/(n*(n-1)))
  inertias.append(kmeans.inertia_)

print("Minimum score: {:.2f}%".format(np.min(scores)))
print("Maximum score: {:.2f}%".format(np.max(scores)))

# Plot normalized criteria.
fig = go.Figure()

fig.add_trace(go.Bar(
  x=list(range(5, 21)),
  y=(scores - np.mean(scores)) / np.std(scores),
  name="Clustering Score"
))

fig.add_trace(go.Bar(
  x=list(range(5, 21)),
  y=(inertias - np.mean(inertias)) / np.std(inertias),
  name="Clustering Inertia"
))

fig.update_layout(
  title="Normalized Clustering Score And Inertia",
  xaxis_title="Condition Number",
  height=800
)
```

```
40
41 fig.show()
42
43 #Minimum score: 81.84%
44 #Maximum score: 94.50%
```

Listing 18: Perform K-means clustering and compare performance using varying number of clusters.

Figure 15: Comparison of K-means performance using varying number of clusters.

As expected from the structure complexity of this dataset, simply looking at these criteria doesn't give any clue on the correct number of clusters that one should use. When K varies from 5 to 20, the inertia decreases: this is expected, since adding more centroids will decrease the sum of distances of samples to their closest cluster center. On the other hand, the clustering score increases way past the $K = 10$ mark, without giving any clue on the correct number of clusters.

Finally, we observed that K-means clustering alone wasn't able to accurately find

meaningful clusters or subclusters in the data. To improve its performance, it should be applied after a dimensionality reduction step such as PCA or MDS.

3.2 Hierarchical clustering

In this section, we will study a new clustering method: agglomerative hierarchical clustering. In particular, we are interested in the effect of different linkage methods on the resulting clusters:

- **ward** minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the K-means objective function but tackled with an agglomerative hierarchical approach;
- **maximum or complete linkage** minimizes the maximum distance between observations of pairs of clusters;
- **average linkage** minimizes the average of the distances between all observations of pairs of clusters;
- **single linkage** minimizes the distance between the closest observations of pairs of clusters.

Listing 19 selects 25 samples of each of the first three digits at random and performs the clustering using different linkage methods. Figure 16 to 19 show the respective clustering results using dendrograms.

```
# Select 25 samples of each of the first three digits.
n_samples = 25
n_digits = 3
samples = np.zeros((n_digits, n_samples, m))
for i in range(n_digits):
  idx = np.random.choice(np.where(y == i)[0], size=n_samples, replace
     =False)
  samples[i] = X[idx]

# Reshape samples into a single array.
data_array = np.reshape(samples, (n_digits*n_samples, m))
labels = [0]*25 + [1]*25 + [2]*25

# Plot dendrogram for ward linkage.
fig1 = ff.create_dendrogram(
  data_array,
  labels=labels,
  linkagefun=lambda x: linkage(data_array, method="ward", metric="
     euclidean"),
  orientation="bottom",
  color_threshold=130
)

```

```
fig1.update_layout(title="Ward Linkage Dendrogram",
  xaxis_title="True Digit",
  yaxis_title="Distance Between Children"
)

fig1.show()

# Plot dendrogram for average linkage.
fig2 = ff.create_dendrogram(
  data_array,
  labels=labels,
  linkagefun=lambda x: linkage(data_array, method="average", metric="
     euclidean"),
  orientation="bottom",
  color_threshold=40
)

fig2.update_layout(title="Average Linkage Dendrogram",
  xaxis_title="True Digit",
  yaxis_title="Distance Between Children"
)

fig2.show()

# Plot dendrogram for complete linkage.
fig3 = ff.create_dendrogram(
  data_array,
  labels=labels,
  linkagefun=lambda x: linkage(data_array, method="complete", metric=
     "euclidean"),
  orientation="bottom",
  color_threshold=48
)

fig3.update_layout(title="Complete Linkage Dendrogram",
  xaxis_title="True Digit",
  yaxis_title="Distance Between Children"
)

fig3.show()

# Plot dendrogram for single linkage.
fig4 = ff.create_dendrogram(
  data_array,
  labels=labels,
  linkagefun=lambda x: linkage(data_array, method="single", metric="
     euclidean"),
  orientation="bottom",
  color_threshold=29
)

fig4.update_layout(
```

```
  title="Single Linkage Dendrogram",
  xaxis_title="True Digit",
  yaxis_title="Distance Between Children"
)

fig4.show()
```

Listing 19: Perform hierarchical clustering and display dendrograms.

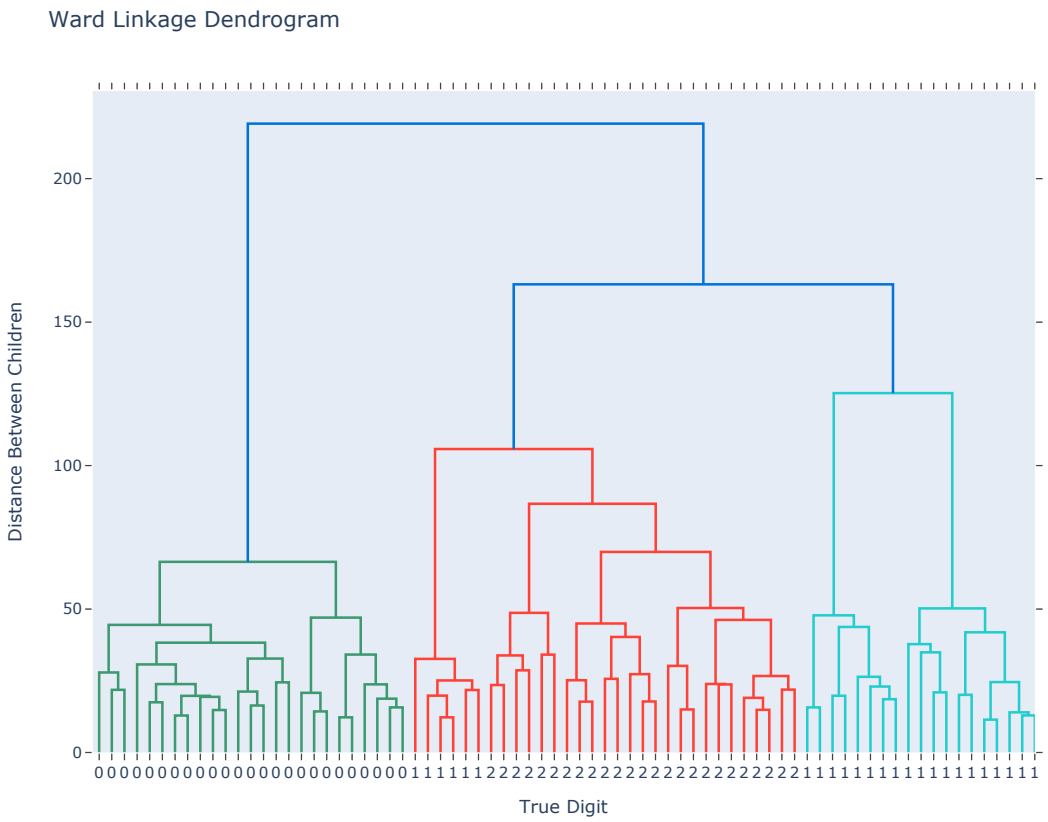


Figure 16: Dendrogram of ward linkage clustering.

The color thresholds were fixed to identify meaningful clusters. We observe on the dendrograms that for all linkage methods, the zeros are almost perfectly clustered, in particular using single and ward linkage. However, the ones and twos show more overlap: only the ward linkage shows almost perfect clustering, while the average linkage displays minimal overlap. Single and complete linkage aren't able to cluster these digits in a meaningful way. This overlap between ones and twos is consistent with the mis-grouping observed on the *t*-SNE visualization: this could indicate that there is some interpenetration of these digits in the raw dataset.

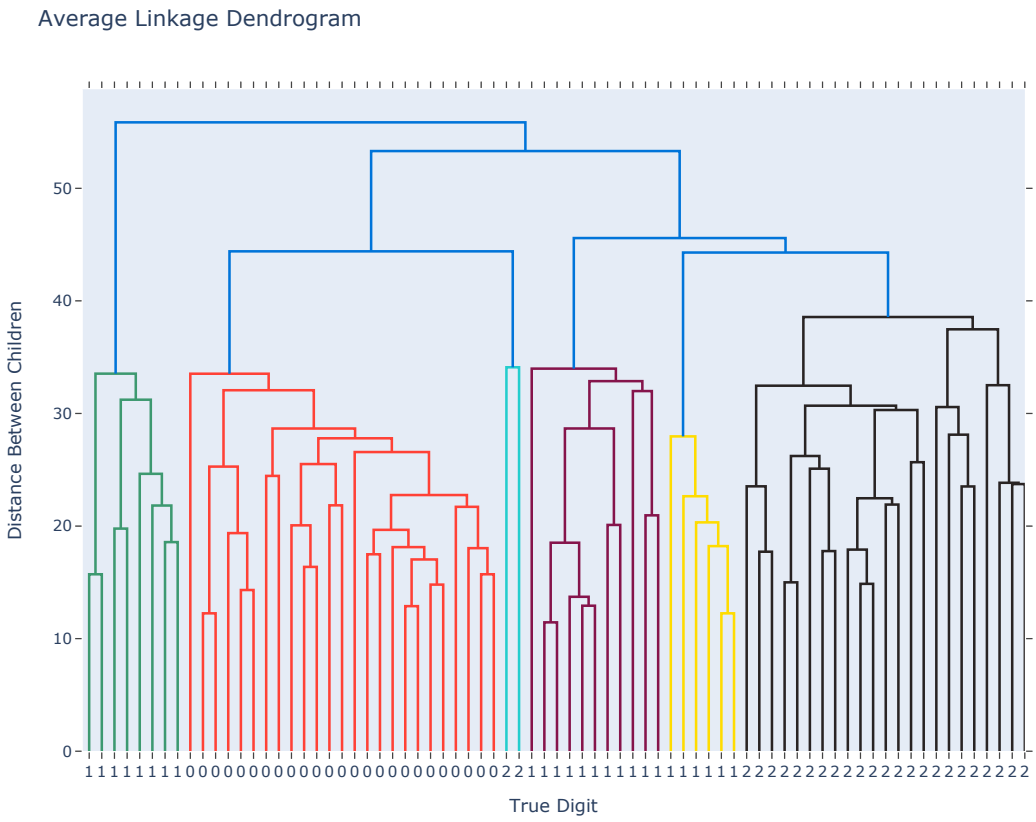


Figure 17: Dendrogram of average linkage clustering.

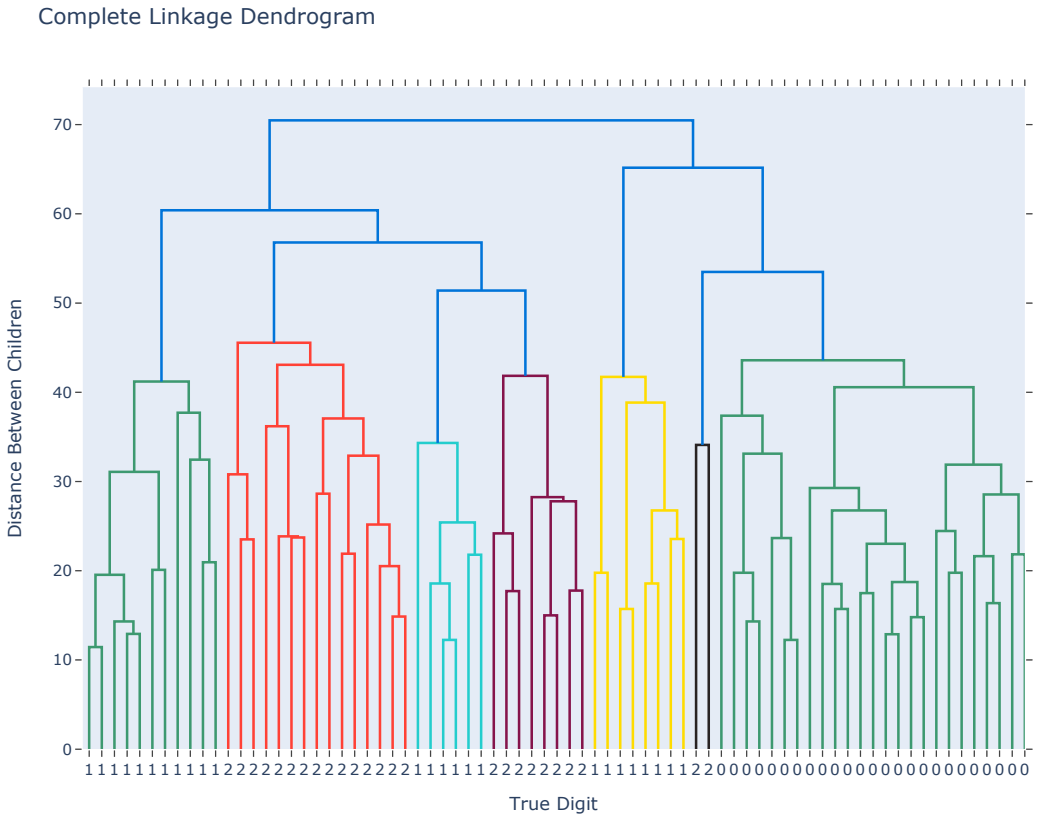


Figure 18: Dendrogram of complete linkage clustering.

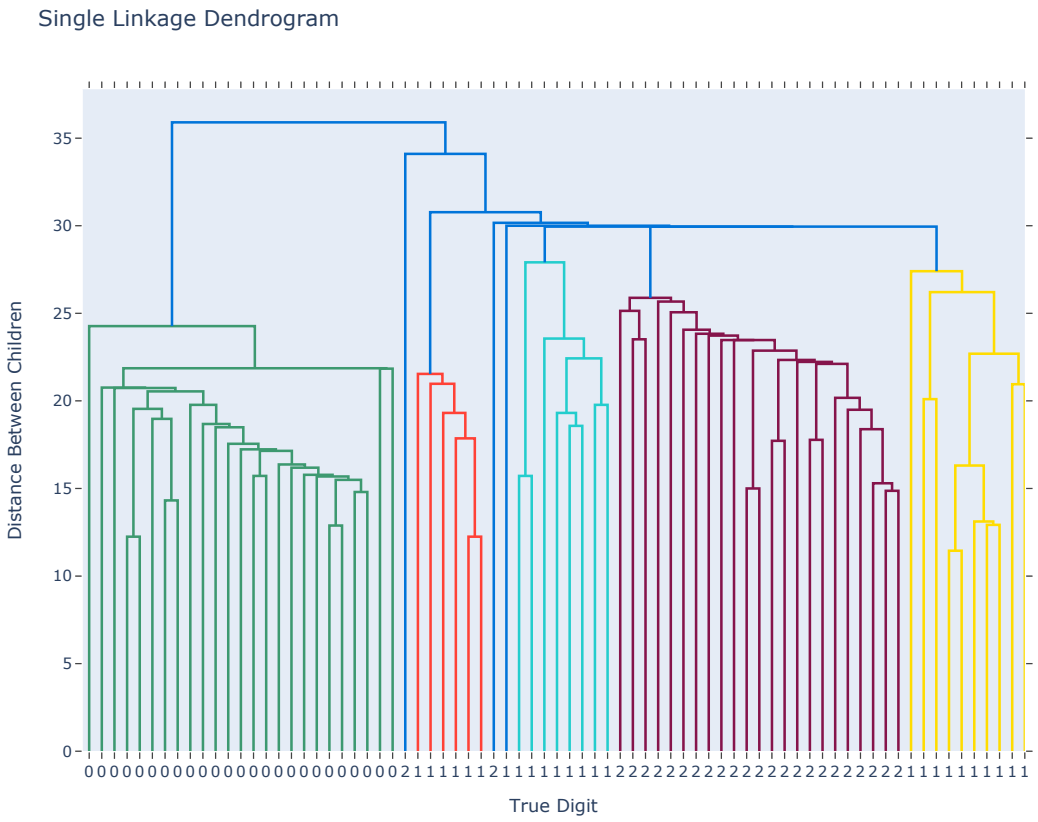


Figure 19: Dendrogram of single linkage clustering.