Assignment 5-6 - Introduction to Data Science

Ninell Oldenburg

April 7, 2022

Exercise 1 - Bayesian Statistics

a) $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x)^2/2\sigma^2}$

b) The conditional probability distribution is $y_n \sim \mathcal{N}(0, \sigma^2)$ because we know that $\varepsilon \sim \mathcal{N}(0, \sigma^2)$

c) The parameters are α, the intercept, and β, the slope. σ, however, depends on the the ε.

d) Priors for α, β, σ, but of course you can put as many priors on priors as you want.

You can use a Gaussian distribution for α, β, but it doesn't make sense mathematical sense for σ to be negative. To avoid that, you could square σ over a half Gaussian.

Exercise 2 - Using Linear Regression

b)

Parameter name	value
w_0	5.21
1. fixed acidity	0.05

Table 1: Parameter names and their value after performing multivariate linear regression on the first column of wine data set

The parameter w_0 is the start parameter. It's value seems reasonable since we want to arrive at a value lying between 0 and 10. The second parameter, which is the first parameter of the data set, seems to have a relatively low contribution to the overall score.

c)

Performing the linear regression on all parameters of the data set, brings us to interestingly different results in contrast to only performing it on the first one, that we can see in Table 2.

First of all, we can observe that Density is contributing in that amount to the overall thing whereas this and that w_0 is about 10 times as high as in exercise b). This can be an indicator for the high influence of some of the parameters as we adjust the weights later on.

Furthermore, it seems that the only parameter we've been including in b), the fixed acidity, has now an even smaller influence on the overall outcome. That makes sense since we're not expecting this weight to increase when we take more, potential influencing parameters into account.

All the other parameters seem to have an almost equally small influence in comparison to the density, that really sticks out with a value of -47.21, which accounts for almost 95% of the $w_0 = 51.66$. We can conclude, that a very large amount of the overall wine quality is defined by the density.

Parameter name	value
w_0	51.66
1. fixed acidity	0.02
2. volatile acidity	-1.06
3. citric acid	0.03
4. residual sugar	0.05
5. chlorides	-2.75
6. free sulfur dioxide	0.01
7. total sulfur dioxide	-0.004
8. density	-47.21
9. pH	-0.43
10. sulfates	0.85
11. alcohol	0.24

Table 2: Parameter names and their value after performing multivariate linear regression on the wine data set

Exercise 3 - Root Mean Square Error

b)

`RMSE of first parameter "fixed acidity": 0.79`.

c)

`RMSE of all parameters: 0.64`. As expected, the RMSE is smaller in comparison to if we only use one parameter for the linear regression. Interestingly, the RMSE is still comparably high.

Exercise 4 - Random forest & normalization

Tree-based classifiers are not affected by standardization. The reason for this is, that the output class is not defined by any distance measures but rather by the class that is predicted by the most trees. Would we scale the features, the split between each of the variables would still be the same.

Exercise 5 - Applying random forest

`Accuracy of sklearn.ensemble.RandomForestClassifier on IDS Weed Crop data: 0.967`

Exercise 6 - Gradient descent & learning rates

```
Function value for learning rate 0.1 = (inf) after 69 iterations
Function value for learning rate 0.01 = (0.993826848) after 55 iterations
Function value for learning rate 0.001 = (0.9938268722) after 485 iterations
Function value for learning rate 0.0001 = (0.9938293111) after 3747 iterations
```

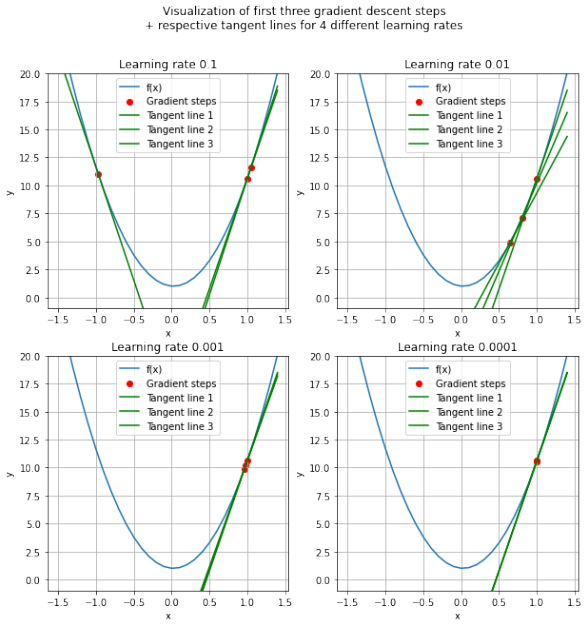

Figure 1: Visualization of first three gradient descent steps + respective tangent lines for 4 different learning rates

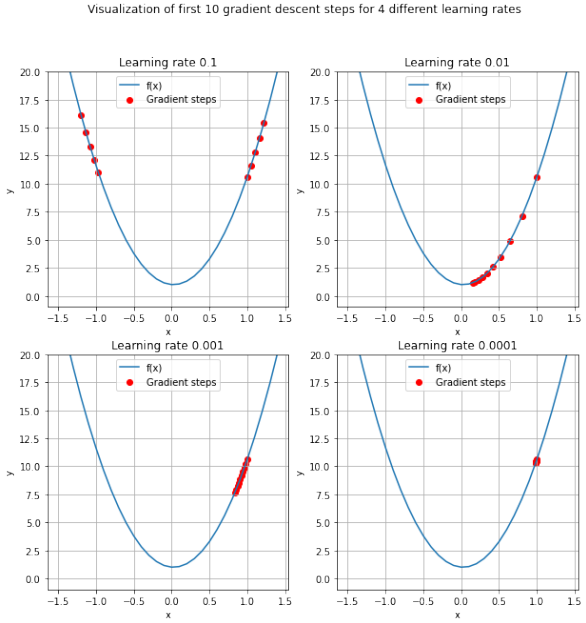

Figure 2: Visualization of first 10 gradient descent steps for 4 different learning rates

Exercise 7 - Logistic regression implementation

1.

We can see that the data points can be clustered comparatively well with human judgement, this is, the two iris classes can be separated into two clusters.

Yet, the data in the 2D2 data set seems to be even better separable than in the 2D1, which leads me to conclude that Iris virginica and Iris versicolor are better distinguishable from each other than Iris virginica and Iris setosa, based on the length of their sepals and, maybe, width of their sepals or length of their petals. But here, I do not want to conclude something as the latter parameters are only

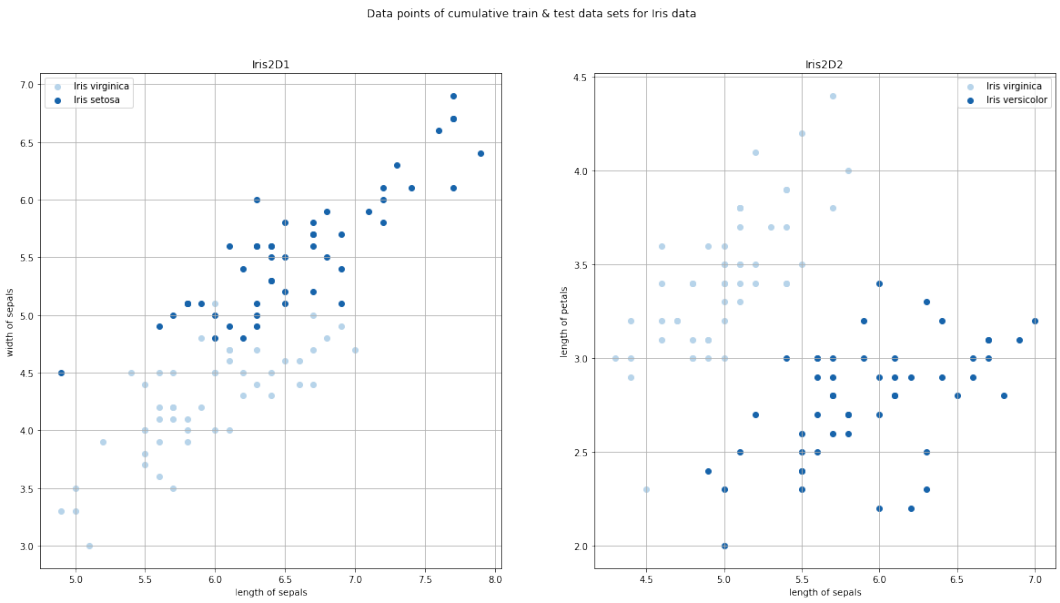

Figure 3: Data points of cumulative train & test data sets for Iris data

given for one of the data sets, respectively.

Lastly, it seems that length and width of sepals correlate linearily uniform accross virginica and setosa, whereas length of sepals and length of petals correlate more parallel throughout virginica and versicolor.

4.

```
Logistic regression for Iris 2D2:
Train error rate:  0.014
Test error rate:  0.0
Parameters:  [-4.28764413 5.99937161 -9.15906681]
Logistic regression for Iris 2D1:
Train error rate:  0.071
Test error rate:  0.1
Parameters:  [-7.34272577 -5.10230049 7.93059724]
```

Exercise 8 - Logistic regression loss-gradient

1.

$$E_{\text{in}}(w) \overset{\text{from lecture}}{=} \frac{1}{N}\sum_{n=1}^{N}\ln\left(1+e^{-y_n w^\top x_n}\right)$$

$$\nabla E_{in}(w) = \frac{\partial E_{in}(w)}{\partial w}\left(\frac{1}{N}\sum_{n=1}^{N}\ln\left(1+e^{-y_n w^\top x_n}\right)\right)$$

$$= \frac{1}{N}\sum_{n=1}^{N}\frac{1}{1+e^{-y_n w^\top x_n}}(-y_n x_n \cdot e^{-y_n w^\top x_n})$$

$$= -\frac{1}{N}\sum_{n=1}^{N} y_n x_n \frac{1}{1+e^{-y_n w^\top x_n}} \cdot 1$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\frac{y_n x_n}{1+e^{-y_n w^\top x_n}}$$

2.

We have the function $E_{\text{in}}(w) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n w^\top x_n}\right)$ and we only look at the part $e^{-y_n w^\top x_n}$ that describes the error.

If label y_n and risk score $w^\top x_n$ agree, meaning they are both positive or both negative, we end up with a large exponent and overall a very small number for $e^{-y_n w^\top x_n}$ and therefore very small contribution to the error.

However, if there is disagreement, this is if y_n is e.g. negative and risk score $w^\top x_n$ is very high, the result of $-y_n w^\top x_n$ would be a positive exponential and therefore the error would become larger.

Overall, this makes a lot of sense since we want to adjust the model whenever we see new, misclassified evidence, but want to keep the weights and parameters whenever the predictions are correct.

Exercise 9 - Clustering and classification I

a)

Description of the software. The multidimensional scaling function is taken from my assignment 3 and takes as input a data set and the number of desired output dimensions. It works through performing a PCA on the data, slicing the eigenvectors in respect to the input value for the dimensions, and outputting the dot product, this is the projection, of the data and its PC eigenvectors. Furthermore, I've added a one liner to output the cumulative variance at this point as this is often used afterwards.

The PCA function is also taken from assignment 3, takes as input a data set and outputs the eigenvalues, eigenvectors and the mean of the data. I start by first centring the data to stabilize the output PC1, compute the covariance matrix, and output the performed the eigen-decomposition as well as the mean (I decided to just output it at this point as it will often be used from the calling functions).

For the clustering part, I've used the `KMeans` clustering method provided by the `sklearn.cluster` package. Here is how it works:

1. **Create a `KMeans` Class**. Here, we pass the parameters that we want to specialize our `KMeans` cluster with. In general, all of the parameters, e.g. the number of target clusters, the initial cluster centers, the maximum number of iterations to find cluster center, etc. have a default value, so we don't *have* to specify these. In our case, however, we tell the algorithm that we want to have 2 clusters, that it should start at `random_state=42`, we want it to use the `"full"` algorithm (so force it to not optimize on the more memory intense algorithm if the data clusters end up being well defined), that we only want to have one iteration with different center seeds (which makes sense as we are initializing the centers ourselves and increasing that number wouldn't change the output), and finally, pass the two center seeds that we've defined before as suggested by the assignment.

   The algorithm then returns a K-means class that is tuned with these parameters and is ready to fit our data.

   For this assignment, we fit the data to the projection of the PCs onto the original data. In this way, we get cluster centers for the whole data with respect to the first x PCs.

2. **Fit our data**. This is done in several stages.

   (a) **Choose initial cluster centers**. That can be done in several ways. 1) Centers are being initialized by the calling function (via the `init=` parameter). In sake of robustness and because we already give the function a `random_state` here (explained in the following), we leave it out for our purposes. The other way is the 2) the random or semi-random initialization where the centers are either being assigned completely random or semi-random by also selecting a `random_state` to make the results reproducible. The number of initial cluster centers corresponds of course always to the number of cluster centers we want to have and pass through the calling function (`n_clusters` parameter).

   (b) **Assign data points to closest centers**. Assign every data point (by which I mean vectors, not every single point!) to their closest center. So in our case every of the 784 points in our

900 vectors is being assigned to their respective closest points at the respective index. In our case there are only two options.

(c) **Calculate new cluster centers**. Now, that all data points are assigned to either of the clusters, we calculate the mean over all points in the cluster. That's how the initial cluster centers are being drawn more and more to the actual center of the potential clusters.

(d) **Repeat steps (b) and (c)**. We stop when we've either 1) reached the `max_iter` parameter that is 300 per default, or 2) when the cluster center do not change anymore from the one iteration to the next one (that's how we know we've found the best possible mean for the center seeds of this iteration).

(e) **Compare different cluster seeds**. This step is being skipped in our implementation due to us assigning the initial seeds. But usually, if they're picked at random, it makes sense to compare several seeds and see which one is having the best possible outcome. That is, which distribution of variance of the cluster centers has the most equal one over the data. If the variance over all cluster centers is comparably similar, we know that these are the best centers.

3. **Return attributes**. The `KMeans` class that is now tuned and trained on our data has now several attributes that are interesting for us. For this assignment, we're interested in the `kmeans.cluster_centers_` as well as the `kmeans.labels_`, that give us an array of the size of our data. But instead of the data points/vectors, we get the labels (so the cluster index the point belongs to (in our case either 0, 1, or 2)). Lastly, I'm prepending the true labels to my `kmeans.labels_` to see, which true labels were appended to which cluster. I print out the `Counter()` over these pairs to see, which true labels correspond to which class the most and how they change with increasing number of PCs.

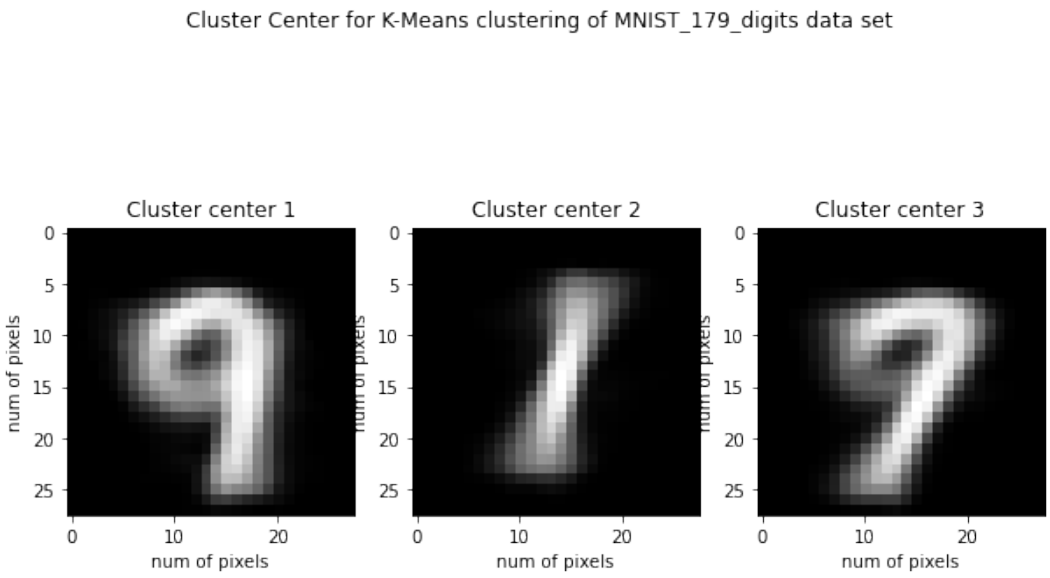

Figure 4: Cluster Center for K-Means clustering of MNIST_179_digits data set

Digits	1	7	9
Cluster			
0	0.3	25.3	**56**
1	**99.3**	10.7	3.3
2	0.3	**64**	40.6

Table 3: Percentages of digits in clusters calculated per hand by taking the `Counter()` output from the code.

b)

```
Accuracy for MNIST_179_digits data set:  0.964 with k_best=1
```

Exercise 10 - Clustering and classification after dimensionality reduction

a)

Description of the software. The multidimensional scaling function is taken from my assignment 3 and takes as input a data set and the number of desired output dimensions. It works through performing a PCA on the data, slicing the eigenvectors in respect to the input value for the dimensions, and outputting the dot product, this is the projection, of the data and its PC eigenvectors. Furthermore, I've added a one liner to output the cumulative variance at this point as this is often used afterwards.

The PCA function is also taken from assignment 3, takes as input a data set and outputs the eigenvalues, eigenvectors and the mean of the data. I start by first centring the data to stabilize the output PC1, compute the covariance matrix, and output the performed the eigen-decomposition as well as the mean (I decided to just output it at this point as it will often be used from the calling functions).

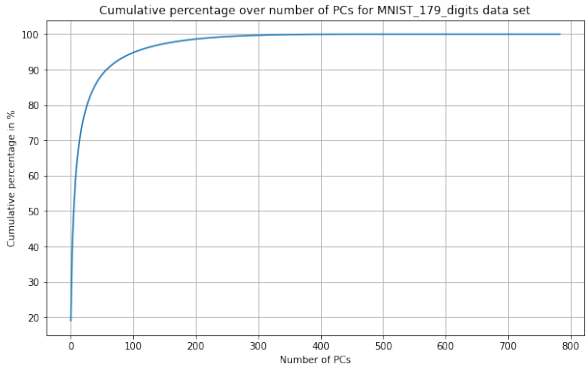

Figure 5: Cumulative percentage over number of PCs for MNIST_179_digits data set

b)

I've started to plot the cluster centers without adding the mean to the projection. We can clearly see an increase in the quality of the projected centers. Following the first three entries in `Counter((1.0, 1.0): 295, (9.0, 0.0): 215, (7.0, 2.0): 140)` of true label and clustered label for this projection, center 0 corresponds to digit 9, 2 to 7 and 1 to 1.

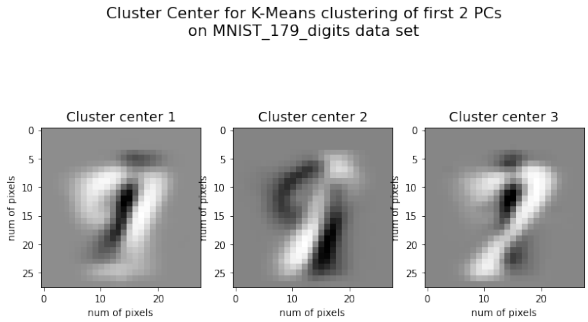

Figure 6: Cluster Center for K-Means clustering of first 2 PCs without added mean

For the first 2 PCs, it seems (again, being displayed without the added mean) that other digits are being displayed in black while the class defining the digit is being displayed in white. In the original data set, the digits are also in white, so it makes sense that the cluster centers have the same color. The result could be considered somewhat surprising as the cumulative variance for the first 2 PCs is 28.56%, which firstly doesn't sound too much, but we have to take into account that this is also computed over the mostly uniform background (where we don't expect a lot of variance to happen).
`Cumulative Variance of first 2 PCs:  28.56%`

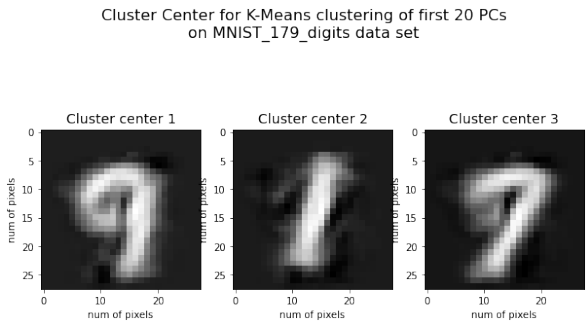

Figure 7: Cluster Center for K-Means clustering of first 20 PCs without added mean

With the growing number of PCs, the cluster centers get more and more clear. Also, the `Counter((1.0, 1.0):  297, (9.0, 0.0):  179, (7.0, 2.0):  179)` increases slightly in the amount of examples for each class in comparison to having only 2 PCs. Especially for the digits 9 and 7, we can observe a higher number of correct classified examples now, which leads to the conclusion that, for now, the digit 1 can better distinguished by the first 2 PCs while 9 and 7 also rely on the following components. As shown below, we are now able to catch 75.19% of the variance, which reflects in the plots.
`Cumulative Variance of first 20 PCs:  75.19%`

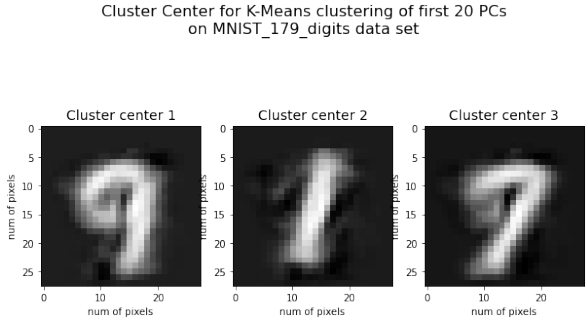

Figure 8: Cluster Center for K-Means clustering of first 200 PCs without added mean

With 200 PCs we should be able to catch 98.61%. This also reflects in the plot: the digits are clearly distinguishable. The `Counter((1.0, 1.0):  298, (7.0, 2.0):  192, (9.0, 0.0):  168)` interestingly decreased for digit 9 while confusing many records with cluster 1, the cluster with primarily records of digit 1, which leads to the conclusion that the more pixels we take into account, the harder it gets to distinguish between 9 and 1.
`Cumulative Variance of first 200 PCs:  98.61`

Below, I've also plotted to pictures for the added mean. But for the sake of showing the influence of different PCs, I've decided to show the cluster centers without the mean for the main description part. But as it seems to be requested from the assignment, please find the plots below.

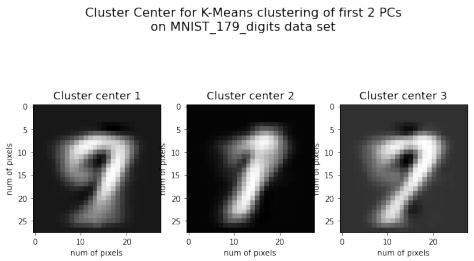

Figure 9: Cluster Center for K-Means clustering of first 2 PCs with added mean

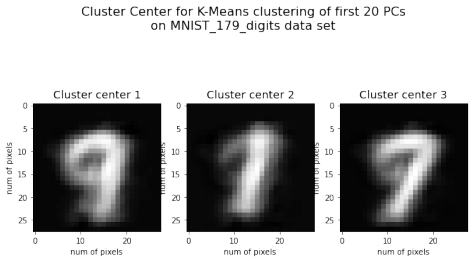

Figure 10: Cluster Center for K-Means clustering of first 20 PCs with added mean

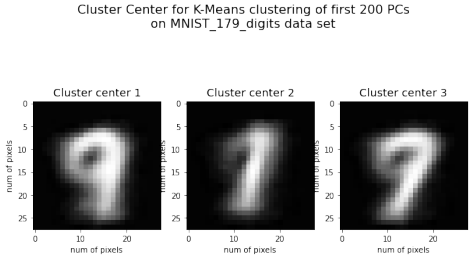

Figure 11: Cluster Center for K-Means clustering of first 200 PCs with added mean

c)

```
Accuracy for MNIST_179_digits data with train data for the first 20 PCs:
   0.2622 with k_best=1
Accuracy for MNIST_179_digits data with train data for the first 200 PCs:
   0.2311 with k_best=1
```

I was in doubt about my results in 10c) as they are clearly not as good as the 0.974 accuracy as in 9b). The reason for this could be that, as shown in 10b), we are only able to catch this high of a percentage for the respective number of PCs. However, what led me to conclude that something is wrong is the fact, that the accuracy decreases with increasing number of PCs. That shouldn't be. However, I wasn't able to find the error in my data and left it for my own research for after the assignment deadline.