Homework 3: Advanced Deep Learning and Latent Variable Models

Reno Malanga, Christian Arndt
Behavioral Data Science

April 4, 2025

Problem 1

Mark all statements which are FALSE.

- In a convolutional neural network (CNN), suppose we replace all 3×3 filters with 1×1 filters while keeping the number of filters and layers unchanged. This transformation does not affect the receptive field of the network.
- The sum of Shapley values for all features in a given instance equals the difference between the model's prediction for that instance and the model's average prediction over the entire dataset.
- A model is said to be intrinsically interpretable if its decision-making process can be directly understood without additional post-hoc methods. Given this definition, decision trees are always more interpretable than models distilled from deep neural networks.
- In model distillation, the student model can never outperform the teacher model in terms of both accuracy and interpretability simultaneously.
- The covariance matrix is not well-defined when $D > N$; that is, there are more features than data points.
- Latent variable models marginalize out the data, $\int p(x, z \mid \theta) dx$, to specify a simpler model via the likelihood of latent variables $p(z \mid \theta)$.
- Factor analysis (FA) assumes a Gaussian prior over latent variables.
- Principle components analysis (PCA) assumes a Gaussian prior over principle components.

- PCA relies on the eigendecomposition of the data matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ where each row is a feature vector of dimension D.
- Results from PCA and FA are always interchangeable.

Problem 2

Convolutional Neural Networks

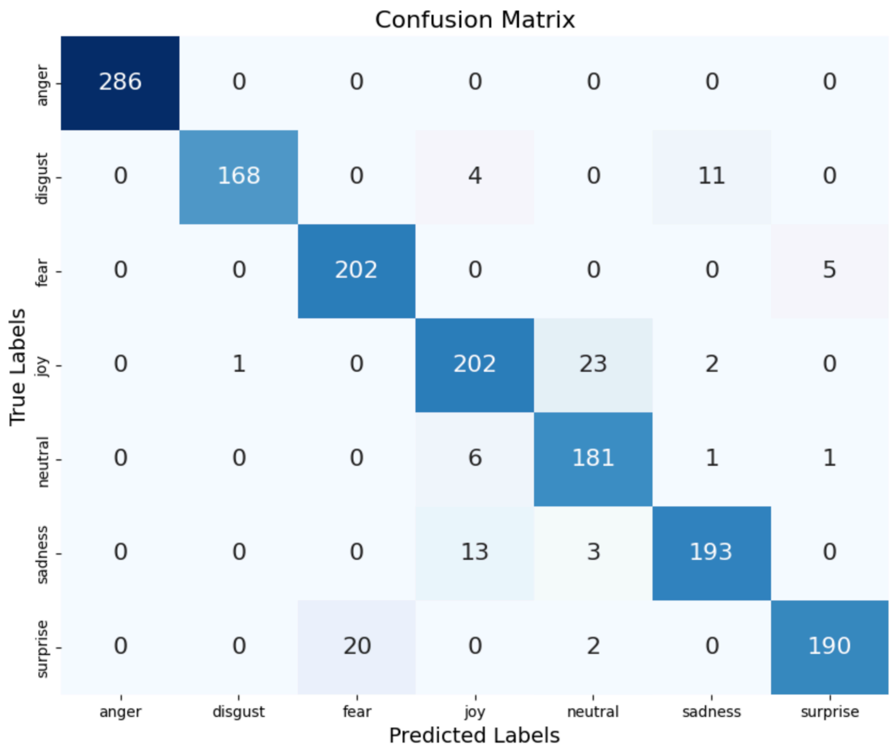
Problem Setup

Create a convolutional neural network (CNN) classifier for emotion recognition using a subset of the FERG database, which is available on LMS under Week 3. Use the provided helper notebook to get started with data loaders. The latter are often necessary for large data sets that cannot fit directly in memory.

1. Load and preprocess the dataset:
   - Use the provided dataset and explore the images and labels.
   - Normalize the images and ensure they are in a suitable format for training.
2. Build a CNN model (either TensorFlow or PyTorch is acceptable):
   - Use a convolutional, pooling, and fully connected layers.
   - Choose an appropriate activation function (e.g. ReLU for hidden layers, softmax for output)
3. Train and evaluate the model:
   - Split the data into training and validation sets.
   - Use an appropriate loss function (**hint**: categorical cross-entropy) and optimizer
   - Monitor training performance and avoid overfitting using dropout.
4. Test the model:
   - Evaluate the model on a test set and analyze its performance.
   - Report accuracy and confusion matrix for model evaluation.

Problem Solution

Our model has an accuracy of about $.94$. Below is a confusion matrix based on our model's predictions:



We could have used a 2 x 2 confusion matrix to show how many type 1 and type 2 errors the model would have made. However, a major downside of this approach would be its inability to demonstrate where the model makes specific mistakes (as in, which emotion-pairs have a higher likelihood of being confused). Based on this 7 x 7 confusion matrix, we see that the majority of incorrect predictions reside in misclassifying joy as neutral and surprise as fear. Seeing this is not unexpected because of the similarity of the two facial expressions.

Problem 3

Shapley Ratios and Interpretability

Problem Setup

First, explain the key differences between permutation importance and Shapley ratios for explaining the predictions of ML models. Next, explore the DeepExplainer from the `shap` library:

- `https://shap.readthedocs.io/en/latest/generated/shap.DeepExplainer.html`

Finally, apply the DeepExplainer either to a deep neural network trained on the MARP dataset or a new network trained on the simpler California data set, which can be loaded like this:

- `X, y = shap.datasets.california(n_points=1000)`

Interpret the results. Which are the three most important features and how do they affect the predictions?

Problem Solution

Differences between Shapley Values and Permutation Importances

Permutation importances have one main problem: because they only look at a single feature at a time, they're insensitive to correlated features. Put another way: when two feature are highly correlated, permuting one of those features will not necessarily create outputs that properly reflect the permuted feature's importance, because the correlated feature can 'pick up the slack', so to speak.

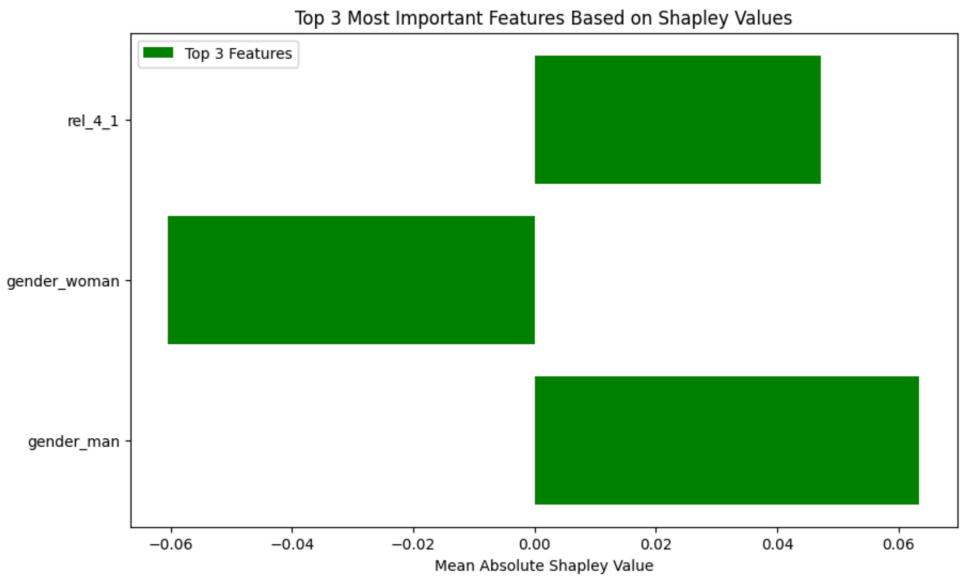
Shapley values get around this by considering each feature along with *every other possible subset of features* (called 'coalitions', as inherited from game theory). This makes Shapley values expensive to compute directly, but far more robust as true feature importance metrics.

Shapley on MARP

In the python notebook `question_3.ipynb`, we:

- load and prepare the MARP dataset,
- train a deep neural network to predict the overall well-being score based on the selected features,
- initialize a `shap.DeepExplainer` on a random selection of 100 datapoints,
- extract Shapley value estimates based on a random selection of 1000 data points, and finally
- average the Shapley values across the data points for each feature, giving us an estimate of the feature's true Shapley value

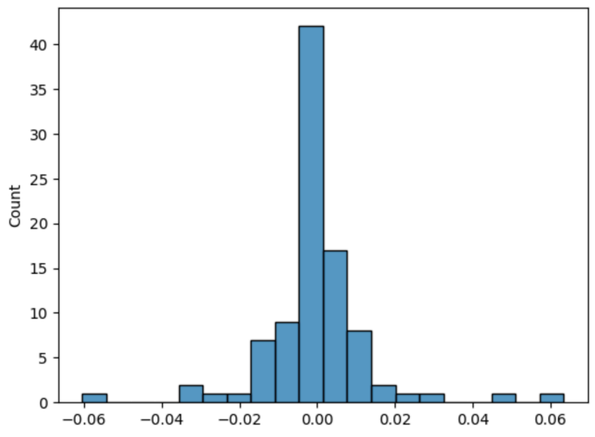
We then sort these averages by absolute value to report the most important features:



It turns out that none of the features have a very high absolute Shapley value. The most important three features are:

1. whether or not the participant is male, with a Shapley value estimate of $\sim .063$
2. whether or not the participant is female, with a Shapley value estimate of $\sim -.060$
3. whether or not the participant belongs to a denomination, with a Shapey value estimiate of $\sim .047$

The signs of the Shapley values tell us the type of correlation the feature has with the output: a positive Shapley value estimate indicates a direct correlation, and a negative Shapley value estimate indicates a negative correlation. Still, all of the Shapley value estimates are incredibly close to 0. In fact, if we plot a histogram of the Shapley value estimates, we can see that the vast majority of features have basically no significant contribution to the outcome of the regression:



The best interpretation we can think of for these results is that none of the features are particularly predictive. This fits in with what we saw in class: it's very difficult to find any significant relationships between the features and the well-being averages in the MARP dataset.

Problem 4

Maximum Likelihood Estimation Continued

Problem Setup

In the previous homework, you explored the connection between maximum likelihood estimation (MLE) and information theory through the Kullback-Leibler divergence. In this problem, you will derive two of the most commonly used loss functions in deep learning: *mean squared error* (MSE) and *binary cross-entropy* (BCE). By applying MLE, you will understand how these losses naturally arise from probabilistic assumptions about a model's outputs.

The MSE and BCE loss functions are defined as follows:

$$MSE(\theta) = \frac{1}{N}\sum_{n=1}^{N}(y_n - f(\mathbf{X}_n;\theta))^2 \tag{1}$$

$$BCE(\theta) = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log(f(\mathbf{x}_n;\theta)) + (1-y_n)\log(1-f(\mathbf{x}_n;\theta))] \tag{2}$$

where:

- N is the number of data points
- $y_n \in \mathbb{R}$ in eq. 1, representing continuous targets for regression
- $y_n \in \{0, 1\}$ in eq. 2, representing binary class labels for classification
- $f(\mathbf{x}_n;\theta)$ represents the model's predicted output for input vector $\mathbf{x}_n$. In eq. 2, the output is bounded between 0 and 1 via an appropriate activation function (e.g., sigmoid).

Your task is to derive the MSE and BCE loss functions by assuming:

- A Gaussian likelihood for the MSE loss (regression):

$$y_n \sim \mathcal{N}(y_n \mid f(\mathbf{x}_n;\theta), \sigma^2)$$

- A Bernoulli likelihood for the BCE loss (classification):

$$y_n \sim \text{Bernoulli}(f(\mathbf{x}_n;\theta))$$

Hint: Instead of setting derivatives to zero (which is not feasible due to the non-convex nature of deep networks), differentiate the respective likelihood functions with respect to the network parameters to derive the corresponding loss functions.

Bonus problem: Derive a loss function for regression where the variance σ^2 is not fixed but is also predicted by the network. (**Hint**: this is known as heteroskedastic loss.)

Problem Solution

MLE and Mean Squared Error

Let us first consider MSE. As stated above:

$$y_n \sim \mathcal{N}(f(\mathbf{x}_n, \sigma^2))$$

From this, we have the PDF of y_n:

$$p(y_n) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n - f(\mathbf{x}_n;\theta))^2}{2\sigma^2}\right)$$

We can plug the PDF into the MLE formula as follows, and then use algebra and log rules to simplify:

$$\begin{aligned}
\theta_{\text{MLE}} &= \underset{\theta}{\text{argmax}} \left[\sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - f(\mathbf{x}_i;\theta))^2}{2\sigma^2}\right)\right)\right] \\
&= \underset{\theta}{\text{argmax}} \left[\sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(\exp\left(-\frac{(y_i - f(\mathbf{x}_i;\theta))^2}{2\sigma^2}\right)\right)\right] \\
&= \underset{\theta}{\text{argmax}} \left[\sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + -\frac{(y_i - f(\mathbf{x}_i;\theta))^2}{2\sigma^2}\right] \\
&= \underset{\theta}{\text{argmax}} \left[\sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + -\frac{1}{2\sigma^2}(y_i - f(\mathbf{x}_i;\theta))^2\right]
\end{aligned}$$

Now, let us drop any terms and coefficients not dependent on θ, understanding that these terms and coefficients have no bearing on the arg max. We can then move the negative sign outside the sum.

$$\begin{aligned}
\theta_{\text{MLE}} &= \underset{\theta}{\text{argmax}} \left[\sum_{i=1}^{N} -(y_i - f(\mathbf{x}_i;\theta))^2\right] \\
&= \underset{\theta}{\text{argmax}} \left[-\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i;\theta))^2\right]
\end{aligned}$$

Finally, let's take advantage of the fact that maximizing a function is equivalent to a minimizing the negative of that function. With this, we can turn the arg max into an arg min and get rid of the negative sign. While we're at it, let's add the normalizing factor, $\frac{1}{N}$, back into the mix; because this doesn't depend on θ, using it will have no impact on the arg max calculation. It will, however, get us to an equation that looks more like the MSE loss function.

$$\begin{aligned}\theta_{\text{MLE}} &= \underset{\theta}{\text{argmax}}\left[-\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i;\theta))^2\right]\\ &= \underset{\theta}{\text{argmin}}\left[\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i;\theta))^2\right]\\ &= \underset{\theta}{\text{argmin}}\left[\frac{1}{N}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i;\theta))^2\right]\end{aligned}$$

We can see that, in the final line above, the formula inside the arg max is exactly the right side of the MSE loss function as given in the problem. Having arrived here from simple algebraic manipulation of the MLE formula, we can see that using MLE over θ is equivalent to minimizing MSE loss function (at least, when we assume a normal likelihood) over θ.

MLE and Binary Cross-Entropy

Let us now consider MSE. As stated above:

$$y_n \sim \text{Bernoulli}(f(\mathbf{x}_n, \sigma^2))$$

From this, we have the PMF of y_n:

$$p(y_n) = f(\mathbf{x}_n;\theta)^{y_n}(1 - f(\mathbf{x}_n;\theta))^{1-y_n}$$

Now, we can use the same process as above: plug the PMF into the formula for MLE and then use algebra and log rules to try to nudge the resulting equation into something that looks like the binary cross-entropy loss function.

$$\begin{aligned}\theta_{\text{MLE}} &= \underset{\theta}{\text{argmax}}\left[\sum_{i=1}^{N}\log\left(f(\mathbf{x}_i;\theta)^{y_i}(1 - f(\mathbf{x}_i;\theta))^{1-y_i}\right)\right]\\ &= \underset{\theta}{\text{argmax}}\left[\sum_{i=1}^{N}\log\left(f(\mathbf{x}_i;\theta)^{y_i}\right) + \log\left((1 - f(\mathbf{x}_i;\theta))^{1-y_i}\right)\right]\\ &= \underset{\theta}{\text{argmax}}\left[\sum_{i=1}^{N}y_i\log\left(f(\mathbf{x}_i;\theta)\right) + (1-y_i)\log\left((1 - f(\mathbf{x}_i;\theta))\right)\right]\end{aligned}$$

As before, let's add the normalizing coefficient of $\frac{1}{N}$ back into the mix:

$$\theta_{\text{MLE}} = \underset{\theta}{\operatorname{argmax}} \left[\frac{1}{N} \sum_{i=1}^{N} y_i \log\left(f(\mathbf{x}_i;\theta)\right) + (1-y_i)\log\left((1-f(\mathbf{x}_i;\theta))\right)\right]$$

Finally, let's turn the maximum into a minimum and add the requisite negative sign:

$$\theta_{\text{MLE}} = \underset{\theta}{\operatorname{argmin}} \left[-\frac{1}{N} \sum_{i=1}^{N} y_i \log\left(f(\mathbf{x}_i;\theta)\right) + (1-y_i)\log\left((1-f(\mathbf{x}_i;\theta))\right)\right]$$

Once again, we've reached a point where the function inside the arg min is identical to the BCE loss function. This means that, when we assume a bernoulli likelihood, finding the model parameters which minimize a binary cross-entropy loss function is equivalent to using maximum likelihood estimation.

Bonus: MLE-Derived Loss Function for a Heteroskedastic Model

Let us now consider the bonus question: what kind of loss function will we get if we assume a normal likelihood as before, but don't assume a known standard deviation? Since our model is now responsible for predicting both the mean and standard deviation of y_n, it is now $\mathbb{R}^D \to \mathbb{R}^2$; that is, it produces a 2-dimensional output. This is a perfectly valid model form, but we need to be able to refer independently to the mean and standard deviation in order to write the likelihood PDF, so let us define two new functions:

- $\boldsymbol{\mu}(\mathbf{x}_n;\theta)$, defined as $f(\mathbf{x}_n;\theta)$ projected onto the μ axis, and
- $\boldsymbol{\sigma}(\mathbf{x}_n;\theta)$, defined as $f(\mathbf{x}_n;\theta)$ projected onto the σ axis

The symbols are bolded to differentiate them from non-functions.

We can plug these functions into the PDF for a Gaussian distribution to get the following:

$$p(y_n) = \frac{1}{\sqrt{2\pi\boldsymbol{\sigma}^2(\mathbf{x}_n;\theta)}} \exp\left(-\frac{(y_n - \boldsymbol{\mu}(\mathbf{x}_n;\theta))^2}{2\boldsymbol{\sigma}^2(\mathbf{x}_n;\theta)}\right)$$

Let us plug the above into the MLE formula as before and do some simplifications:

$$\begin{aligned}
\theta_{\text{MLE}} &= \underset{\theta}{\text{argmax}} \left[\sum_{i=1}^{N} \log \left(\frac{1}{\sqrt{2\pi\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta)}} \exp\left(-\frac{(y_i - \boldsymbol{\mu}(\mathbf{x}_i;\theta))^2}{2\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta)} \right) \right) \right] \\
&= \underset{\theta}{\text{argmax}} \left[\sum_{i=1}^{N} \log \left(\frac{1}{\sqrt{2\pi\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta)}} \right) + \log \left(\exp\left(-\frac{(y_i - \boldsymbol{\mu}(\mathbf{x}_i;\theta))^2}{2\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta)} \right) \right) \right] \\
&= \underset{\theta}{\text{argmax}} \left[\sum_{i=1}^{N} -\frac{1}{2} \log \left(2\pi\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta) \right) - \frac{(y_i - \boldsymbol{\mu}(\mathbf{x}_i;\theta))^2}{2\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta)} \right] \\
&= \underset{\theta}{\text{argmax}} \left[\sum_{i=1}^{N} -\frac{1}{2} \log (2\pi) - \frac{1}{2} \log \left(\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta) \right) - \frac{1}{2\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta)} (y_i - \boldsymbol{\mu}(\mathbf{x}_i;\theta))^2 \right] \\
&= \underset{\theta}{\text{argmax}} \left[-\frac{1}{2} \sum_{i=1}^{N} \log (2\pi) + \log \left(\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta) \right) + \frac{1}{\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta)} (y_i - \boldsymbol{\mu}(\mathbf{x}_i;\theta))^2 \right]
\end{aligned}$$

Once again, we can drop any terms unrelated to θ along with the coefficient of $\frac{1}{2}$ without changing the result of the arg max:

$$\theta_{\text{MLE}} = \underset{\theta}{\text{argmax}} \left[-\sum_{i=1}^{N} \log \left(\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta) \right) + \frac{1}{\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta)} (y_i - \boldsymbol{\mu}(\mathbf{x}_i;\theta))^2 \right]$$

Finally, let's flip the sign and turn the likelihood maximization into a minimization problem. Also, for the sake of consistency, let's add the normalizing term, $\frac{1}{N}$, back in.

$$\theta_{\text{MLE}} = \underset{\theta}{\text{argmin}} \left[\frac{1}{N} \sum_{i=1}^{N} \log \left(\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta) \right) + \frac{1}{\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta)} (y_i - \boldsymbol{\mu}(\mathbf{x}_i;\theta))^2 \right]$$

Repeating the pattern from the first two cases, the function inside the arg min is our loss function:

$$\mathcal{L}_{\text{heteroskedastic}}(\mathbf{X};\theta) = \frac{1}{N} \sum_{i=1}^{N} \log \left(\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta) \right) + \frac{1}{\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta)} (y_i - \boldsymbol{\mu}(\mathbf{x}_i;\theta))^2$$

We don't recognize this function (or any algebraic equivalent we've tweaked this function into) as a common form, but it seems like it makes sense as a loss function. To wrap things up, here are a few remarks about it:

- The MSE loss function is contained within this heteroskedastic loss function, normalized with the inverse variance. This makes intuitive sense: high MSE is still punished via loss, but if our model also predicts a high variance, the loss punishment is reduced. In other words, if our model expects the data to have a large spread, it punishes deviations from the mean less.

- One important feature of a good loss function is that there are no trivial solutions to minimization. In other words, our model shouldn't be able to 'game' the variance component of the loss. The above loss function has no trivial solutions; this can be seen by looking at the possible extremes our model could predict:

  - As the variance prediction of our model gets closer to 0, the $\log(\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta))$ term approaches $-\infty$, but the variance-dependent coefficient of the MSE term approaches ∞. We can determine whether the limit converges or, if not, how it diverges by taking a limit. To simplify the math, let's consider the MSE as a constant c. In this case, the loss function simplifies to:
  $$\log(\boldsymbol{\sigma}^2(\mathbf{x}_n;\theta)) + \frac{c}{\boldsymbol{\sigma}^2(\mathbf{x}_n;\theta)}$$
  Now, we can take a limit as the variance approaches 0:
  $$\lim_{\boldsymbol{\sigma}^2(\mathbf{x}_n;\theta)\to 0+} \left[\log(\boldsymbol{\sigma}^2(\mathbf{x}_n;\theta)) + \frac{c}{\boldsymbol{\sigma}^2(\mathbf{x}_n;\theta)}\right] = \infty$$
  (proof by WolframAlpha). From this, we can see that, regardless of our model's mean error, our model cannot trivially minimize loss by predicting a small variance.
  - As the variance prediction of our model gets closer to ∞, the $\frac{1}{\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta)}(y_i - \boldsymbol{\mu}(\mathbf{x}_i;\theta))^2$ term disappears, but the $\log(\boldsymbol{\sigma}^2(\mathbf{x}_i;\theta))$ blows up. In other words:
  $$\lim_{\boldsymbol{\sigma}^2(\mathbf{x}_n;\theta)\to \infty} \left[\log(\boldsymbol{\sigma}^2(\mathbf{x}_n;\theta)) + \frac{c}{\boldsymbol{\sigma}^2(\mathbf{x}_n;\theta)}\right] = \infty$$
  This means that we cannot minimize loss by predicting an arbitrarily large variance, either.

- The loss has a direct relationship with the variance. This makes sense; a high variance indicates a model unable to extract much information from the training data, which is obviously undesirable. A good model should predict as low a variance as it can given the data.

- The loss scales with the log of the variance, while it scales with the square of the mean absolute error. This means that, while the loss will punish a high variance, it will punish a high mean absolute error much more vigorously. We aren't sure if there is a more rigorous way to put this, but such behavior seems good for a loss function.

  One way to look at it is this: the variance can be thought of as a sort of confidence measure for our model's predictions, while the mean is just the prediction itself. An ideal model should be confident (i.e., low variance), so it makes sense that the loss scales with variance; but more importantly, an ideal model should predict the correct values, which is why it makes sense that the actual prediction error (i.e., mean absolute error) has a quadratic relationship with the loss. Put another way, this loss function will prefer a model which predicts accurately with high confidence most of all, but if one of accuracy or confidence must be sacrificed, it will prefer a model which predicts accurately with low confidence over a model which predicts inaccurately with a high confidence.

Problem 5

Factor Analysis and PCA

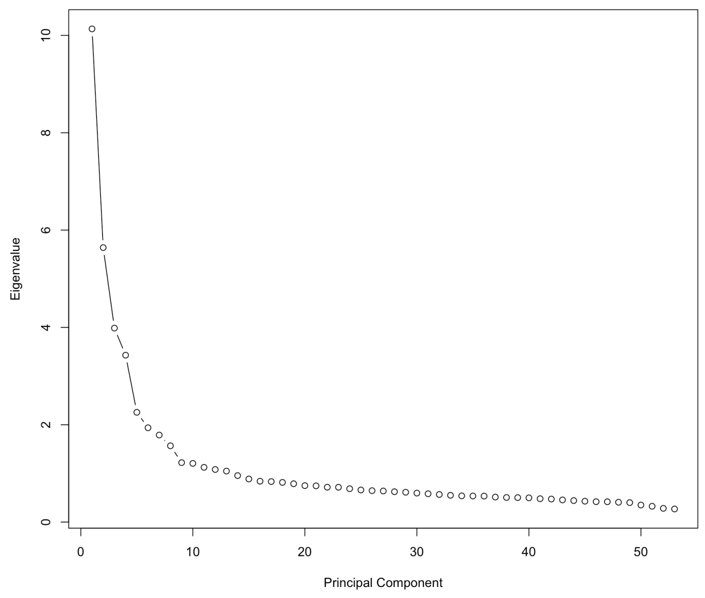
Problem Setup

Perform a complete factor analysis (FA) on the provided personality dataset. For simplicity, use only five subscales of your choice. Your analysis should include:

- Data cleaning, missing value imputation, and exploratory analysis (e.g., heatmaps)
- A well-reasoned decision on the number of factors to extract
- Computation of **loadings** and **communalities**, along with their interpretation
- Evaluation of the subscale structure. Do items belonging to the same subscale load highly on a single factor while showing minimal loadings on others? Which factor explains the most variance? Which items should be removed (e.g. due to low loadings), if any?

Problem Solution

We chose the first five subscales from the personality data for this assignment, which include fields A, B, C, D, and E. After choosing the subscales and performing a factor analysis on the data, we used a scree plot to determine the number of important factors. This plot is shown below.



Based on the plot, we chose to do a five-factor analysis. This was a somewhat vibes-based decision based on the two heuristics we learned in class; we were initially torn between using 3 and 5 factors. Our thinking was as follows:

- There is a bit of an elbow at the third eigenvalue, but the fifth eigenvalue has a much more obvious elbow.
- The fifth eigenvalue is also well above the cutoff threshold of 1.

With both of these facts in mind, we decided that a five-factor analysis might be more descriptive than a three-factor one.

Having performed the analysis, we report the loadings and communalities by subscale below. Long story short, the factor analysis does a poor job of re-creating the data's structure. A look at the loadings makes it obvious that it's possible to identify which factor picked up which subscale: subscale A corresponds to factor 2, B to 4, C to 1, D to 3, and E to 5. That saiide, the loadings themselves are rather low. The majority of loadings are below 0.6, which leads to the explained variability of these fields generally being less than 0.4.

The communalities are the sum of the squared factors, which allows us to determine how much each variable's variance is explained by the factors. In doing so, we have found that the majority of the communalities are less than 0.5. This means that most of the data is not well explained by the factors.

Loadings by Subscale

	F1	F2	F3	F4	F5
A1	-0.12	0.67	0.11	0.04	0.12
A2	-0.13	0.63	0.12	0.03	0.29
A3	0.09	0.65	-0.03	0.01	0.01
A4	-0.02	0.63	0.00	0.02	0.11
A5	-0.17	0.63	0.07	0.05	0.28
A6	-0.22	0.61	0.09	0.08	0.15
A7	-0.08	0.63	0.01	-0.01	0.05
A8	-0.04	-0.37	-0.05	-0.02	-0.08
A9	0.14	-0.58	0.04	-0.04	-0.10
A10	0.09	-0.35	-0.00	-0.08	0.07

	F1	F2	F3	F4	F5
B1	0.01	0.14	0.13	0.53	-0.04
B2	0.04	-0.07	0.17	0.54	0.01
B3	0.07	0.08	0.15	0.43	-0.16
B4	-0.13	0.11	0.16	0.56	-0.12
B5	-0.12	0.07	0.18	0.52	-0.04
B6	0.11	-0.10	0.24	0.42	0.11
B7	-0.13	0.13	0.00	0.24	-0.31
B8	-0.14	0.15	0.13	0.27	-0.14
B9	0.07	0.11	-0.19	-0.33	-0.02
B10	0.42	0.01	-0.17	-0.41	0.09
B11	0.30	-0.05	-0.24	-0.19	-0.07
B12	0.10	-0.03	-0.01	-0.57	0.06
B13	0.10	0.03	0.03	-0.47	0.07

	F1	F2	F3	F4	F5
C1	-0.48	0.05	0.08	-0.05	0.02
C2	-0.62	0.12	0.22	0.09	0.04
C3	-0.42	0.12	0.26	0.22	0.03
C4	-0.50	0.09	-0.05	0.06	0.16
C5	-0.47	0.12	-0.05	0.05	-0.04
C6	0.65	-0.04	0.03	-0.08	0.07
C7	0.76	-0.08	-0.11	0.02	-0.07
C8	0.71	-0.10	-0.21	-0.02	-0.06
C9	0.70	-0.06	-0.13	-0.08	-0.01
C10	0.61	-0.02	-0.23	-0.21	-0.05

	F1	F2	F3	F4	F5
D1	-0.16	0.17	0.79	0.12	0.06
D2	0.04	0.06	0.65	0.09	0.13
D3	-0.10	0.04	0.35	0.21	0.23
D4	-0.10	-0.09	0.35	0.28	0.12
D5	-0.13	0.14	0.73	0.12	-0.00
D6	-0.10	0.05	0.37	0.30	0.10
D7	0.27	-0.05	-0.59	-0.17	-0.08
D8	0.14	-0.03	-0.28	-0.37	-0.08
D9	0.19	0.04	-0.54	-0.14	-0.03
D10	0.41	0.11	-0.36	-0.19	-0.08

	F1	F2	F3	F4	F5
E1	-0.17	0.23	0.29	-0.05	0.62
E2	-0.16	0.20	0.19	-0.14	0.70
E3	0.03	0.22	-0.01	0.23	0.49
E4	-0.09	0.18	0.15	-0.20	0.68
E5	-0.05	0.34	0.03	0.28	0.40
E6	0.14	0.09	0.13	-0.03	0.58
E7	0.04	-0.16	0.09	-0.34	-0.22
E8	0.24	-0.16	-0.12	0.15	-0.58
E9	0.16	-0.12	0.03	-0.31	-0.13
E10	0.01	-0.05	0.02	-0.05	-0.40

Communalities

	A	B	C	D	E
1	0.49	0.32	0.24	0.70	0.56
2	0.51	0.33	0.45	0.45	0.62
3	0.43	0.25	0.30	0.23	0.35
4	0.41	0.38	0.29	0.24	0.57
5	0.51	0.33	0.25	0.58	0.35
6	0.45	0.27	0.44	0.25	0.39
7	0.40	0.19	0.60	0.46	0.20
8	0.15	0.15	0.56	0.24	0.46
9	0.37	0.16	0.51	0.35	0.16
10	0.14	0.38	0.47	0.36	0.16
11		0.19			
12		0.34			
13		0.24			