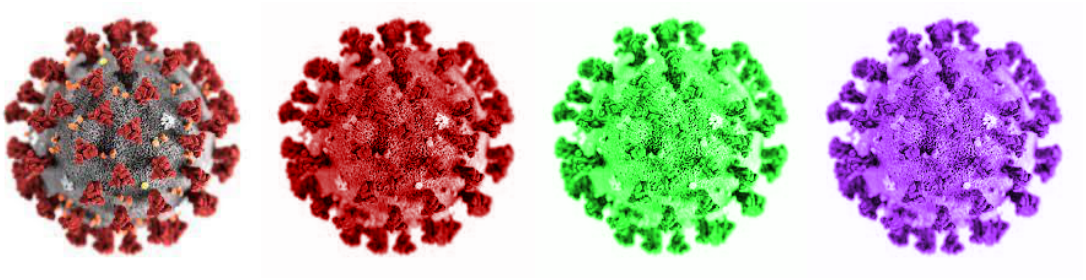
HW3: Regression

In this final assignment, we will try to predict a continuous label using our dataset.

Good Luck!



Instructions

- **Submission**
  - Submit by Tuesday, **02.04.2024**, 23:59.
  - Submissions in pairs only on the webcourse.

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

- **Final report**
  - Should be written in a word processor (Office Word, Google docs, etc.).
    - Should not contain the code itself.
    - Do not submit jupyter notebooks as PDFs.
  - Can be in Hebrew, English, or both.
  - **You are primarily assessed based on your written report.**
  - Answer the questions in this instruction file according to their numbering.
  - Add concise explanations, figures (outputs of your code), tables, etc.
  - You are evaluated for your answers but also for clarity, readability, and aesthetics.
  - **Tables** should include feature names and suitable titles.
  - **Plots** should have suitable titles, axis labels, legends, and grid lines (when applicable).

- **Submit a zip file containing** (please use hyphens, not underscores):
  - Define *<filename>* as your dash-separated IDs, i.e., *id1-id2* or *id1-id2-id3*.
  - The zip file's name should be *<filename>.zip* (e.g., *123456789-200002211.zip*).
  - **Only one group member should submit the assignment to the webcourse!**
  - The report PDF file with all your answers (but not your code!), named *<filename>.pdf*.
  - Your code (choose the relevant options for you):
    - Working with jupyter: a notebook with your code, named *<filename>.ipynb*.
    - Working with a "traditional" IDE: one clear main script, named *<filename>.py*, and any additional files required for running the main script.
  - A completed *LinearRegressor.py* file with your implementation.

Preliminary: Updated Data Loading

Task: Follow the procedure below.

a. Start by **loading** the **new** raw data in *`HW3_data.csv`*.
   The dataset is almost identical to the one from the previous assignments, but we deleted the binary targets and revealed the continuous `contamination_level` target variable.

b. Make sure the data is **partitioned** correctly to train and test, according to the instructions in the previous assignments.
   The train-test partitions **must** be **identical** to the ones you used in HW1 and HW2.

c. Apply the **preprocessing** procedure from the previous assignments (including the normalization steps) on both the training and test sets (but be careful to only use the training set when computing statistics for normalization etc.).

   Note: in Lecture 08 we explain why some preprocessing steps (e.g., normalization), should be applied to the validation folds according to statistics computed on the train fold. Here, for simplicity only, you can compute these statistics according to all the training samples (before splitting it to train and validation folds).

d. **Important:** do not use the test set before Section 6.

Throughout this assignment, we mainly focus on regressing the new continuous `contamination_level` variable.

Section 1: Linear regression implementation

This assignment focuses on linear regression. However, instead of using the MSE loss which yields the least squares formulation that we saw in class, we will focus on minimizing the **Huber** loss, which yields a different optimization problem. Read the definition and motivation parts of the Huber loss Wikipedia page.

The problem we solve is defined as (for a given $\delta \geq 0$):

$$\min_{\boldsymbol{w}\in\mathbb{R}^d,\ b\in\mathbb{R}} \mathcal{L}_H(\boldsymbol{w},b), \quad \text{where } \mathcal{L}_H(\boldsymbol{w},b) = \frac{1}{m}\Sigma_{i=1}^{\mathrm{m}}\ell_\delta(\boldsymbol{w},b;\boldsymbol{x}_i,y_i)$$

and

$$\ell_\delta(\boldsymbol{w},b;\boldsymbol{x}_i,y_i) = \begin{cases} \frac{1}{2}(\boldsymbol{w}^\top\boldsymbol{x}_i+b-y_i)^2, & |\boldsymbol{w}^\top\boldsymbol{x}_i+b-y_i|\leq\delta \\ \delta\left(|\boldsymbol{w}^\top\boldsymbol{x}_i+b-y_i|-\frac{1}{2}\delta\right), & otherwise \end{cases}.$$

The (sub)gradient with respect to the weight vector is:

$$\mathbb{R}^d \ni \nabla_{\boldsymbol{w}}\ell_\delta(\boldsymbol{w},b;\boldsymbol{x}_i,y_i) = \begin{cases} (\boldsymbol{w}^\top\boldsymbol{x}_i+b-y_i)\boldsymbol{x}_i, & |\boldsymbol{w}^\top\boldsymbol{x}_i+b-y_i|\leq\delta \\ \delta\cdot\mathrm{sign}(\boldsymbol{w}^\top\boldsymbol{x}_i+b-y_i)\cdot\boldsymbol{x}_i, & \text{otherwise} \end{cases},$$

Both of the following questions are analytical. You are required to type your derivations on a computer (i.e., with Word's equation tool, LaTeX/LyX, or equivalent tools).

(Q1) In your report, prove that the (sub)derivative of the loss with respect to the bias is:

$$\frac{\partial\ell_\delta(\boldsymbol{w},b;\boldsymbol{x}_i,y_i)}{\partial b} = \begin{cases} (\boldsymbol{w}^\top\boldsymbol{x}_i+b-y_i), & |\boldsymbol{w}^\top\boldsymbol{x}_i+b-y_i|\leq\delta \\ \delta\cdot\mathrm{sign}(|\boldsymbol{w}^\top\boldsymbol{x}_i+b-y_i|), & \text{otherwise} \end{cases}$$

(Q2) In your report, write the analytical gradients $\nabla_{\boldsymbol{w}}\mathcal{L}_H(\boldsymbol{w},b)$, $\frac{\partial\mathcal{L}_H(\boldsymbol{w},b)}{\partial b}$ in vector form (e.g., without an explicit summation $\sum$) for given $\mathbf{X}\in\mathbb{R}^{m\times d}$ and $\boldsymbol{y}\in\mathbb{R}^m$.

For instance, the gradient in the ordinary least squares case that we learned can be written in the two following ways:

$$\mathbb{R}^d \ni \nabla_{\boldsymbol{w}}\mathcal{L}_{\mathrm{LS}}(\boldsymbol{w},b) = \frac{2}{m}\sum_{i=1}^{m}(\boldsymbol{w}^\top\boldsymbol{x}_i+b-y_i)\boldsymbol{x}_i = \underbrace{\frac{2}{m}\mathbf{X}^\top(\mathbf{X}\boldsymbol{w}+b\mathbf{1}_m-\boldsymbol{y})}_{\text{vector form}}$$

(Q3) Given a dataset $\{(\boldsymbol{x}_i,y_i)\}_{i=1}^m$ suggest an algorithm to determine a reasonable value for δ, to be used in the Huber loss. Ideally, your answer should simply be the pseudo-code for your algorithm with some concise description (5-7 sentences maximum) and should describe how your suggested algorithm considers the specific dataset you are handling and how to estimate a suitable value for δ from it.

To implement our regressor we will inherit from sklearn's `BaseEstimator` class (like in HW2) for compatibility with scikit-learn API. We will also inherit from `RegressorMixin`. This is the only section where we will perform validation without using cross validation.

- For **this section only**, split your training set into a (new) training subset (80%) and a validation subset (20%).
- Copy the given `LinearRegressor` module into your notebook / project.
- Complete the following methods.
  - `LinearRegressor.loss` method so that it computes the objective Huber loss $\mathcal{L}_H(\underline{\boldsymbol{w}}, b)$ on a given dataset. Avoid using for loops.
  - `LinearRegressor.gradient` method so as to compute the analytic (vector-form) gradients $\nabla_{\underline{\boldsymbol{w}}}\mathcal{L}_H(\underline{\boldsymbol{w}}, b)$ and $\frac{\partial}{\partial b}\mathcal{L}_H$. Avoid using for loops.

    Tip: When possible, prefer vector operations (e.g., `np.sum`, `np.linalg.norm`).
  - `LinearRegressor.fit_with_logs` method in the module so as to compute the gradients of the current batch and update the parameters accordingly.
  - `LinearRegressor.predict` method so as to compute the model prediction.

Like in HW2, you will now verify the correctness of your implementation for the loss and its gradient by plotting the residuals $\| \underbrace{\nabla_{\underline{\boldsymbol{w}}}\mathcal{L}_H(\underline{\boldsymbol{w}}, b)}_{\text{analytic}} - \underbrace{u_{\delta_w}(\underline{\boldsymbol{w}}, b)}_{\text{numeric}} \|_2$ and $| \underbrace{\frac{\partial}{\partial b}L(\underline{\boldsymbol{w}}, b)}_{\text{analytic}} - \underbrace{u_{\delta_b}(\underline{\boldsymbol{w}}, b)}_{\text{numeric}} |$ as a function of δ_w, δ_b (respectively; over many repeats).

Task: Copy the functions from the given *`verify_gradients.py`* into your notebook / project. Read and understand these functions but do not edit them.

(Q4) Using your preprocessed (and normalized) dataset and `contamination_level` as our target, generate a plot that compares the numerical gradients to the analytical ones, with $\delta = 0.1$ (the hyperparameter of the Huber loss). Do this by running the following:

```
compare_gradients(X_train, y_train, huber_delta, deltas=np.logspace(-7, -2, 9))
```

Important: `X_train` should hold the features of your **normalized** training subset.
`y_train` should hold the `contamination_level` subset training labels.

Attach the plots to your report. No need to discuss them, but make sure that they make sense.

Task: Copy the function given in *test_lr.py* into your notebook / project.

Read and understand the function but do not edit it.

(Q5) We now want to evaluate the effects of different learning rates on the learning procedure. Run the following command that plots a graph of the training and validation losses as a function of the iteration number for different learning rates. In this section use $\delta = \delta_0$ with δ_0 being the value you have derived in (Q3).

```
test_lr(X_train, y_train, X_val, y_val, huber_delta)
```

Important: `X_val` should hold the features of your (preprocessed) validation subset. `y_val` should hold the `contamination_level` subset validation labels.

Note: If your model did not converge with any learning rate, you are allowed to alter the `lr_list` variable (but explain this in your report).

This part should also help you verify your implementation (loss should decay).

Attach the plots to your report, briefly discuss the results and justify the demonstrated behaviors. Which learning rate is "optimal" (briefly explain)?
For the best learning rate, does it make sense to increase the number of gradient steps (instead of the default 1500 steps)? Explain.

(Q6) The Huber loss is commonly used when trying to learn robust regressors. What are the cases in which the "robustness" of the Huber loss can be best exploited? Especially when compared to ordinary least squares (OLS)? Answers should be concise (5-8 sentences maximum).

Now that we have experienced solving and tuning the least squares problem, we are ready for the rest of the assignment… and save the world from Contamination!

Section 2: Evaluation and Baseline

Our general goal is to minimize the generalization loss, that is $\mathbb{E}_{(\boldsymbol{x},y)\sim D}[\ell_\delta(\boldsymbol{w}, b; \boldsymbol{x}, y)]$. As we have learned, in practice, we instead minimize the empirical error $\frac{1}{m}\sum_{i=1}^{m} \ell_\delta\left(\boldsymbol{w}, b; \boldsymbol{x}_i, y_i\right)$ on a training set, and tune hyperparameters using a validation set.

In the rest of this assignment, we use the k-fold cross-validation method for better estimating the generalization error, thus improving the tuning procedure. We will use $k = 5$ folds.

Similar to HW2, we use `sklearn` to perform cross-validation on the (whole) training set to evaluate the performance of models. Unlike our training objective, which is the Huber loss, the scoring metric we will use to actually evaluate our performance on the test is the classical MSE (using `cross_validate`, set `scoring='neg_mean_squared_error'`).

Simplest baseline

We now train a simple DummyRegressor that always predicts the average `contamination_level` of the training set. We will use this regressor throughout the assignment as a baseline to which we will compare the performance of our learned regressors.

(Q7) Create a DummyRegressor. Evaluate its performance using `cross-validation`. In your report, fill in the cross-validated errors of the regressor.

Model	Section	Train Huber Loss	Valid MSE
		Cross validated	
Dummy	2		

Task: Retrain the dummy regressor on the entire training set (= all its samples) and save it for future use (Sec 6).

Basic hyperparameter tuning

A quick reminder of the tuning process for hyperparameters:

The repeated tuning process (for a single parameter) should include:

i. Determining the tested values of the tuned hyperparameter (see numpy.logspace). You need to choose suitable values by yourself that will help you optimize the validation error.

ii. For each value, evaluating a suitable regressor using cross validation ($k = 5$).

iii. Plotting (cross-validated) train and validation errors as a function of the hyperparameter. Consider using semilogx or loglog plots.
Also plot a constant line with the validation error of the dummy regressor.

iv. Reporting the value that yields the optimal validation error and its respective error.

(Q8) Create an instance of your custom `LinearRegressor` and evaluate its performance using `cross-validation`, **remember to tune the LR!** (for δ, use the δ_0 from (Q3) again).
Report the appropriate plots and values detailed above in "the repeated tuning process".
Fill in the cross-validated errors of the regressor yielded by the optimal hyperparameter.

Model	Section	Train Huber Loss	Valid MSE
		Cross validated	
Dummy	2	filled	filled
Linear	2		

(Q9) Had we chosen not to normalize features beforehand, would the training performance of these two models have changed (assume there are no numerical errors)? Explain.

Task: Using the best performing hyperparameter, **retrain** the regressor on the entire training set (= with all its samples) and save it for future use (Sec 6).

Section 3: Ridge linear regression

In regularized linear regression, we need to tune the regularization strength (λ in our notation, `alpha` in sklearn). As mentioned earlier, k-fold cross-validation is performed with the **entire** training set (= all its samples) for the remainder of the experiments.

We'll now learn to predict `contamination_level` using sklearn's HuberRegressor. Make sure your models are non-homogeneous (`fit_intercept=True`). Note that here δ is determined through the parameter `epsilon`. For its value choose again your δ_0 value.

(Q10) Tune the regularization strength λ of the regressor. Follow the repeated tuning process described earlier. Remember to attach the required plot and specify the optimal strength with its validation error.
Remember that plot should have suitable titles, axis labels, grid lines, etc.

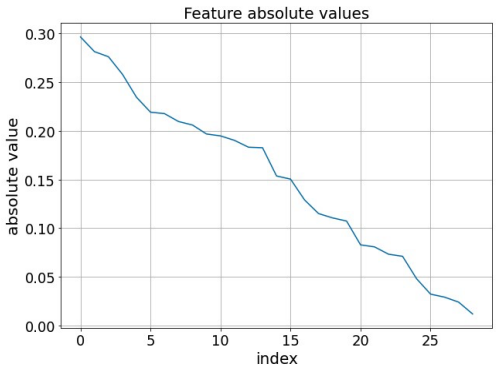
(Q11) Fill in the cross-validated errors of the regressor yielded by the optimal hyperparameter.

Model	Section	Train Huber Loss	Valid MSE
		Cross validated	
Dummy	2	filled	filled
Linear	2	filled	filled
Ridge Linear	3		

Task: Using the best performing hyperparameter, **retrain** the regressor on the entire training set (= with all its samples) and save it for future use (Sec 6).

We now wish to visualize the resulting coefficients.

(Q12) Sort and plot the absolute values of the learned coefficients. The x-axis should be the index of the parameter from largest to smallest.
The result should roughly look like the illustration to the right (the values are made up).



(Q13) Assuming we wish to maximize the interpretability of the importance of each feature to the model, is the Ridge regularization a good choice? If yes, why? If not, which other regularization would you use instead (we recommend you to review the relevant tutorial on regression again before answering)?

Section 4: Feature Mappings (visualization)

As a detour to better understand polynomial fitting for regression, we now focus on regressing the `contamination_level`, using only `PCR_02` and `PCR_06`.

Task: Create a subset of the training set with these 2 features and `contamination_level`.

Task: Visualize the data using `plot3d`, i.e., plot `contamination_level` as a function of the `PCR_02` and `PCR_06` features. (For now, don't use the `predictions` argument.)

(Q14) Attach the 3-d plot to your report. What can we understand from this visualization? How should this affect our choice of model to regress `contamination_level`?

(Q15) We will now train an sklearn.linear_model.HuberRegressor with the optimal regularization strength found in (Q10) (using the two features only), remember again to make sure your models are non-homogeneous (`fit_intercept=True`).

(Q16) We will now visualize the model you just trained. Use `plot3d` to plot all the training samples and their true labels (as before). Also pass your model's predictions in a list to the `predictions` keyword to compare the true labels to the predicted values. Attach the plot to your report.

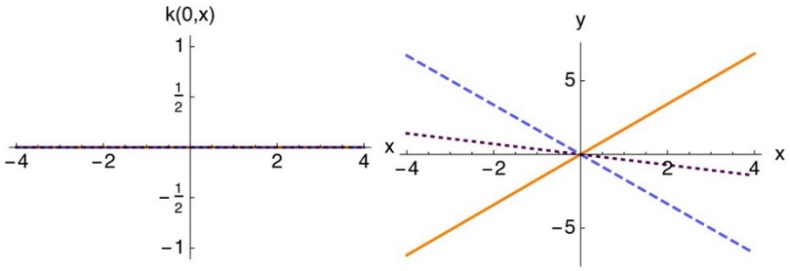
We will now try to improve over the previous model using a "feature mapping".

Task: Create a single model that: (1) transforms the original features using a Laplacian Kernel (2) normalize the transformed features (3) trains a regressor on the normalized transformed features. To ease the usage of the Laplacian kernel, here we will use Scikit's SVM-based regressor: sklearn.svm.SVR. To do so, we will use a Pipeline like we did in HW2. The following snippet creates such a pipeline:

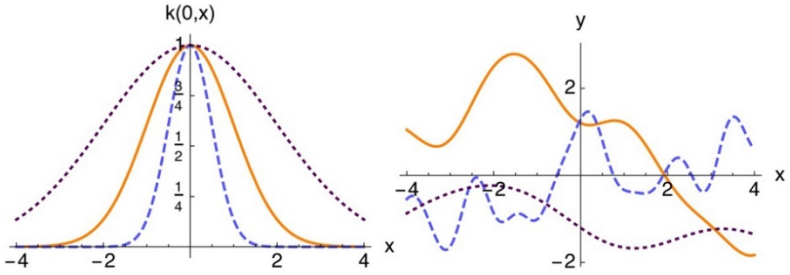
```
poly_reg = Pipeline([(('normalization', TODO),
                      ('regressor',
                      SVR(kernel=sklearn.metrics.pairwise.laplacian_kernel,
                      tol=1e-5, C=C, epsilon=1)
```

Where the regularization term 'C' should be tuned (see next question).

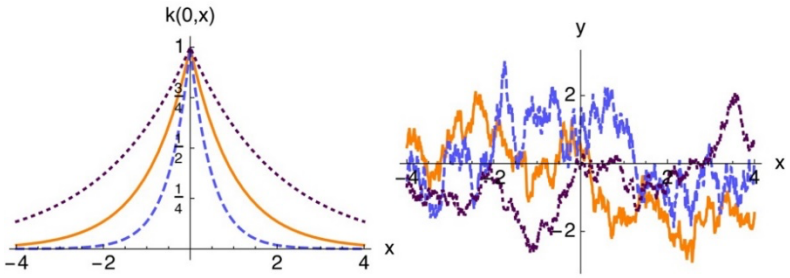
The Laplacian kernel can be thought of as a non-smooth version of Gaussian kernels (not to be confused with the RBF kernel), which serves as a good choice for describing stochastic, non-smooth processes (for example in quantum physics, financial processes, molecular dynamics etc.), for visualization see the following:



(a) Linear kernel.



(b) Gaussian kernel with $\sigma = 0.5, 1, 2$ (dashed, solid, dotted lines).



(c) Laplacian kernel with $\sigma = 0.5, 1, 2$ (dashed, solid, dotted lines).

(Q17) Tune the regularization strength ('C') of the regressor with the Laplacian mapping using cross validation ($k = 5$ as always). (Remember: in HW2 you saw how to set hyperparameters inside a pipeline. Use cross_validate.)

Remember to attach required plots, optimal strengths, and optimal validation errors.

Task: Using the best performing hyperparameter, **retrain** the SVR regressor on the entire training set (= with all its samples).

(Q18) We will now visualize the SVR model you just trained. Use the new `plot3d` function to plot all the training samples and their true labels (as before). Pass your model predictions in a list to the `predictions` keyword to compare the true labels to the predicted values. Attach the plot to your report.

(Q19) Discuss the results from this section. Compare the two models' capacities to fit the data at hand (using the visualizations and metric scores you obtained). This should take 2-4 sentences maximum.

Section 6: Testing your models

Important: do not continue to this section until you have finished all previous sections.

Finally, we can let the **test set** come out and play.

At the end of the previous sections, you retrained the tuned models on the entire training set. You will now evaluate the test errors (using the MSE metric) for the models, as well as the Dummy baseline.

Remember: do not cross-validate here. Train on the entire training set (= using all its samples) and evaluate on the test set.

(Q20) Complete the entire table

Model	Section	Train Loss	Valid MSE	Test MSE
		Cross validated		Retrained
Dummy	2	filled	filled	
Linear	2	filled	filled	
Ridge Linear	3	filled	filled	
SVR + Laplace	4	filled	filled	

Which model performed best on the test set?