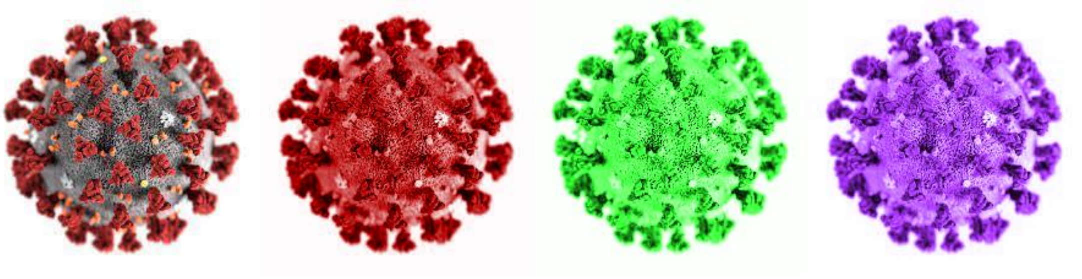
HW3: Regression

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

Good Luck!



Instructions

- **Submission**
    - Submit by Wednesday, **02.02.2025**, 23:59. No late submissions will be accepted.
    - Submissions in pairs only in 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):
    - The report PDF file with all your answers, named *`id1-id2.pdf`*.
    - Your code (choose the relevant options for you):
        - Working with jupyter: a notebook with your code, named *`id1-id2.ipynb`*.
        - Working with a "traditional" IDE: one clear main script, named *`id1-id2.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 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

Before we start using sklearn's modules, we want to thoroughly understand the optimization problem we use to solve linear regression – the least squares problem.
We will now implement a non-homogeneous linear regressor with the stochastic gradient descent (SGD) algorithm. The MSE loss definition for such a regressor is:

$$L(\underline{\boldsymbol{w}}, b) = \frac{1}{m}\sum_{i=1}^{m}\left(\underline{\boldsymbol{w}}^\top \underline{\boldsymbol{x}}_i + b - y_i\right)^2 = \frac{1}{m}\|\boldsymbol{X}\underline{\boldsymbol{w}} + \underline{\mathbf{1}}_m \cdot b - \underline{\boldsymbol{y}}\|_2^2 \,,$$

where $\underline{\mathbf{1}}_m \in \mathbb{R}^m$ is a vector of ones and $b \in \mathbb{R}$ is a scalar. Using $\underline{\mathbf{1}}_m \cdot b$ allows the same b to be used for all rows (corresponding to all samples).
The gradient with respect to $\underline{\boldsymbol{w}} \in \mathbb{R}^d$ was presented in Tutorial 09:

$$\mathbb{R}^d \ni \nabla_{\underline{\boldsymbol{w}}} L(\underline{\boldsymbol{w}}, b) = \frac{1}{m} 2\boldsymbol{X}^\top(\boldsymbol{X}\underline{\boldsymbol{w}} + \underline{\mathbf{1}}_m \cdot b - \underline{\boldsymbol{y}})$$

(Q1) In your report, derive (פַּתח) the analytical partial derivative $\frac{\partial}{\partial b} L(\underline{\boldsymbol{w}}, b) \in \mathbb{R}$.

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 <u>only</u> section where we will perform validation <u>without</u> using cross validation.

- For **<u>this section only</u>**, 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 MSE loss $L(\underline{\boldsymbol{w}}, b)$ on a given dataset. Avoid using for loops.
  - `LinearRegressor.gradient` method so as to compute the analytic gradients $\nabla_{\underline{\boldsymbol{w}}} L(\underline{\boldsymbol{w}}, b)$ and $\frac{\partial}{\partial b} L$. 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 <u>batch</u> and update the parameters accordingly.
  - `LinearRegressor.predict` method so as to compute the model prediction.

You will now verify the correctness of your implementation for the loss and its gradient by plotting the residuals $\| \underbrace{\nabla_{\underline{\boldsymbol{w}}} L(\underline{\boldsymbol{w}}, b)}_{\text{analytic}} - \underbrace{u_{\delta_w}(\underline{\boldsymbol{w}}, b)}_{\text{numeric}} \|_2$ 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.

(Q2) Using your preprocessed (and normalized) dataset and `contamination_level` as our target, generate a plot that compares the numerical gradients to the analytical ones. Do this by running the following command:

```
compare_gradients(X_train, y_train, 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 these plots, 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.

(Q3) 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.

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

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.

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 MSE, that is $\mathbb{E}_{(x,y)\sim D}[(h(x)-y)^2]$.

As we have learned, in practice, we instead minimize the empirical error $\frac{1}{m}\sum_{i=1}^{m}(h(x_i)-y_i)^2$ 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. As explained earlier, the metric we use for regression is the 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.

(Q4) Create a DummyRegressor. Evaluate its performance using `cross-validation`. In your report, fill in the cross-validated errors of the regressor. Note that you need to report the average error across all folds, as evaluated on the 'test' set of each fold, NOT on the actual test set which you should not touch at this stage.

Model	Section	Train MSE	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.

(Q5) Create an instance of your custom `LinearRegressor` and evaluate its performance using `cross-validation`, **remember to tune the LR!**
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 MSE	Valid MSE
		Cross validated	
Dummy	2	filled	filled
Linear	2		

(Q6) 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: Lasso 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 will now learn to predict `contamination_level` from the sklearn.linear_model.Lasso regressor.
Make sure your models are non-homogeneous (`fit_intercept=True`).

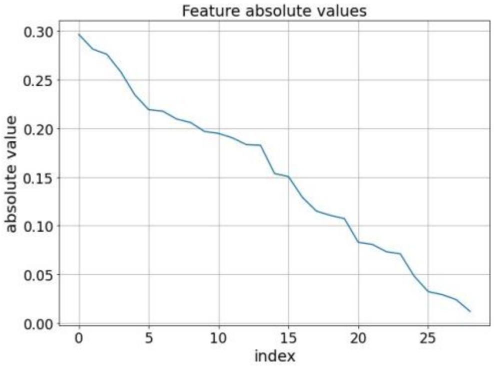
(Q7) Tune the regularization strength of the regressor. Follow the repeated tuning process described earlier. Remember to attach the required plot and specify the optimal strength with its validation error.
Remember that plot should have suitable titles, axis labels, grid lines, etc.

(Q8) Fill in the cross-validated errors of the regressor yielded by the optimal hyperparameter.

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	filled	filled
Linear	2	filled	filled
Lasso 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).

(Q9) Specify the 5 features having the 5 largest coefficients (in absolute value) in the resulting regressor, from the largest to the smallest (among these 5).



We now wish to visualize the resulting coefficients.

(Q10) 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 (in terms of axis and structure, the values her are made up).

(Q11) Why is the magnitude of the coefficients interesting? Explain briefly.

(Q12) Had we chosen not to normalize features beforehand, would the training performance of the Lasso model have changed (assume there are no numerical errors)? Explain.

(Q13) How would you expect the coefficients of the trained model to change, should we have used the Ridge regressor instead? Explain.

Section 4: Polynomial fitting (visualization)

As a detour to better understand polynomial fitting for regression, we now focus on regressing the `contamination_level`, using only `PCR_03` and `PCR_07`.

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_03` and `PCR_07` features. (for now, don't use the `predictions` argument.)

We will now try to improve our linear Lasso-regression model by adding quadratic features.

Task: Create a single model that: (1) transforms the original features into 3rd-degree polynomial features, (2) normalizes the transformed features, and then (3) trains the linear Lasso regressor on these normalized polynomial features. To do so, we will use a Pipeline like we did in HW2. The following snippet creates such a pipeline:

```
poly_reg = Pipeline([('feature_mapping', TODO),
                     ('normalization', TODO),
                     ('Lasso', Lasso(alpha=Lambda, fit_intercept=True)
```

Complete the first two steps in the pipeline above to make it apply a 3rd-degree PolynomialFeatures transformation, followed by a normalization of your choosing (tip: you are allowed to simply apply a `MinMaxScaler` for all the transformed features).

(Q14) Why is re-normalization important after applying a polynomial mapping?
Hint: Your answer should revolve around the degrees of the different monomials and focus on mathematical/"optimizational" considerations rather than on statistical ones like in Lecture 08.

(Q15) Tune the regularization strength of a Lasso regressor with the polynomial mapping using cross validation ($k = 5$ as always). (Remember: on Page 12 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 Lasso regressor on the entire training set (= with all its samples).

(Q16) We will now visualize the polynomial model you just trained. Use the `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.

(Q17) Fill in the train and validation errors of the regressor yielded by the best performing hyperparameter. Remember to compute the errors using cross-validation.

Model	Section	Train MSE	Valid MSE
		Cross validated	
⋮	⋮	filled	filled
Polynomial Lasso	4		

Section 5: Fitting Gradient Boosted Machines (GBM) of the CovidScore

Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models (typically decision trees). When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees (as in the AdaBoost example seen in class). A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

Read here and here for a more in-depth introduction to GBMs, or checkout the wiki page.

For this task, we will train a Scikit's GBM regressor: GradientBoostingRegressor.
Apply the GBM regressor to the following features to the set of most important features as decided by you. Please list the features you have chosen as input.

Task: Create copies of your train and test sets (with all the relevant features).

(Q18) Tune the 'learning_rate', and 'min_samples_leaf' parameters of the GradientBoostingRegressor model using cross-validation with a grid search (see Q8 in Major HW2). For the 'loss' parameter, use 'huber' (for further details regarding the Huber loss see here). Use a similar pipeline to the one created earlier in this section. Remember to attach required heatmaps, optimal hyper-parameters, and optimal train and validation errors (use the default values for all other parameters).

(Q19) Fill in the train and validation errors of the regressor yielded by the best performing hyperparameter. Remember to compute the errors using cross-validation.

Model	Section	Train MSE	Valid MSE
		Cross validated	
⋮	⋮	filled	filled
GBM Regressor	5		

Task: Train the GBM regressor on the entire training set and save it for future use (Sec 6).

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 (MSE) 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 MSE	Valid MSE	Test MSE
		Cross validated		Retrained
Dummy	2	filled	filled	
Linear	2	filled	filled	
Linear Lasso	3	filled	filled	
Polynomial Lasso	4	filled	filled	
GBM	5	filled	filled	

Which model performed best on the test set?

Briefly discuss the results in the table (from an overfitting and underfitting perspective, or any other insightful perspective).