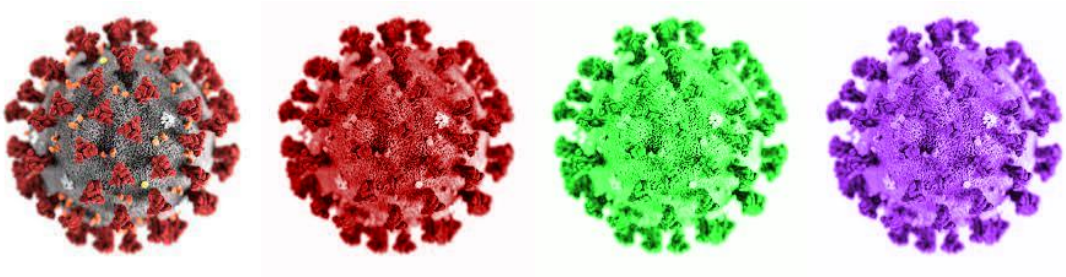
HW3: Regression

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

Scientists have developed an advanced test that computes a probability that a patient is infected with the COVID virus. Unfortunately, this test is costly and cannot be performed at large scale.

Our goal is to regress the outcome of this test, i.e., the “virus score”, with the cheap features we used in the previous assignments.

Good Luck!



Instructions

- **Submission**
  - **Submit by:** Thursday, **27.01.2022**, 23:59.
    - Late submissions will be accepted until 31.01.2022 and penalized 5pts per day.
  - 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).
-
  - **Grading**
    - 10% of the grade is based on your model accuracies and your code:
      - In Section 9 you submit predicted labels for unlabeled data.
      - 10 points are given to all submissions achieving MSEs lower than some thresholds which we do not publish (should be easy to achieve).
    - The other 90% is based on your final report and submitted code.
      Note that some answers are dependent on your personal results. Not all questions have a single answer.
    - A comparative bonus competition with awards of 1-2 points to the final course grade (see Section 8 for details).

- **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.
  - Your predictions files, named *pred_{4,5,6,8}.csv* (see Section 9).

Preliminary: Updated Data Loading

Task: Follow the procedure below.

a. Start by **loading** the **new** raw data in *virus_labeled.csv*.
   The dataset is identical to the one from the previous assignments, but we deleted the binary targets and revealed the continuous `VirusScore` target variable which previously created the binary "`covid`" target variable.

b. Apply the **preprocessing** procedure from the previous assignment (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.).

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

d. **Important:** do not use the test set before Section 7.

e. Make sure you did not delete the following features: `PCR_01`, `PCR_02`, `PCR_03`, `PCR_07`, `PCR_10`, `num_of_siblings`, `sugar_levels`, and `household_income`. Make sure you did not edit/delete the `BloodType` feature. Note that there are other important features for this assignment. Here we chose to specify only some of them. If you deleted some of them, you should edit your preprocessing pipeline accordingly.

f. Make sure you deleted the `Job` feature and binary target variables (`covid`, `risk`, `spread`).

Section 1: Quick data exploration and preparation

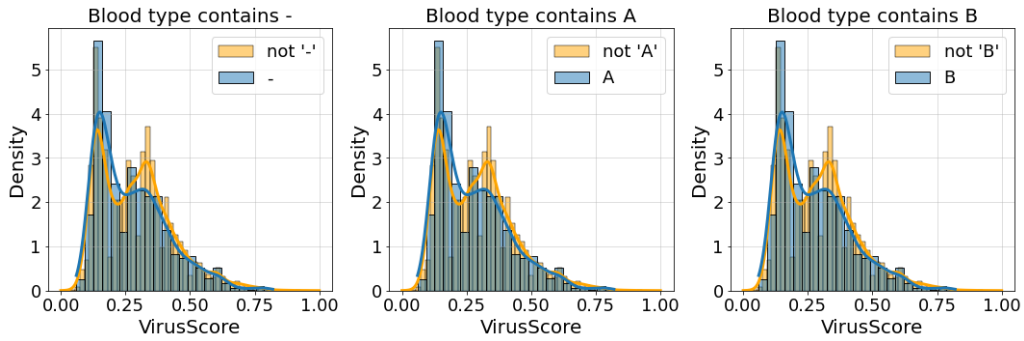
Throughout this assignment, we focus only on regressing the `VirusScore`, which previously created the binary "`covid`" target variable.

The following steps should be done on the training set only.

(Q1) Use `histplot` to plot kde plots of the distributions of the target variable (`VirusScore`) conditioned on the following conditions of `BloodType` of the patient:

i. "+" blood types vs. "-" blood types
ii. "A" blood types vs. "not A" blood types
iii. "B" blood types vs. "not B" blood types

The result should roughly look like the next illustration (the distributions in the illustration are made up).



You can use the following code snippet as a starting point:

```
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))
for i, cur_type in enumerate(['-', 'A', 'B']):
  filter_mask = X_train['blood_type'].str.contains(cur_type)
  sns.histplot(data=y_train[~filter_mask], ax=ax[i], stat="density", kde=True,
               line_kws={"linewidth": 3}, color="orange", label="not '{}'".format(cur_type))
  sns.histplot(data=y_train[filter_mask], ax=ax[i], stat="density", kde=True,
               line_kws={"linewidth": 3}, label=cur_type)
  ax[i].set_title("Blood type contains " + cur_type)
  ax[i].legend(), ax[i].grid(alpha=0.5)
```

Attach the resulting plots to your report.

Remember: make sure to include appropriate title, axis labels, etc.

(Q2) Can you identify a pattern?

Which condition is most informative for learning the `VirusScore` target variable? Explain.

Task: On both the train and test sets, create a **binary** feature for the chosen condition (`1` if it is `True` and `0` if `False`) and drop the `BloodType` feature.

Section 2: Linear regression implementation

Before we start using sklearn's modules, we want to make sure we 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 stochastic gradient descent (SGD) optimization. The MSE loss definition for such a regressor is:

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

where $\underline{\mathbf{1}} \in \mathbb{R}^m$ is a vector of ones and $b \in \mathbb{R}$ is a scalar. Using $\underline{\mathbf{1}} \cdot b$ allows the same scalar b to be used for each row (representing each sample) in the right-hand term.

The gradient with respect to $\underline{\boldsymbol{w}} \in \mathbb{R}^d$ was presented in tutorial 09:

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

(Q3) In your report, derive (חשבו) the analytical partial derivative $\frac{\partial}{\partial b}\mathcal{L}(\underline{\boldsymbol{w}}, b) \in \mathbb{R}$.

To implement our custom regressor, similarly to HW2, we will inherit from the `BaseEstimator` class from sklearn for compatibility with scikit-learn API. We will also inherit from the `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 and a validation subset with a ratio of $80\% - 20\%$.
- Copy the `LinearRegressor` module from the given *`LinearRegressor.py`* into your notebook / project.
- Complete the following methods
  - `LinearRegressor.loss` method so that it computes the objective MSE loss $\mathcal{L}(\underline{\boldsymbol{w}}, b)$ on a given dataset. Avoid using for loops.
  - `LinearRegressor.gradient` method as to compute the analytic gradients $\nabla_{\underline{\boldsymbol{w}}}\mathcal{L}(\underline{\boldsymbol{w}}, b)$ and $\frac{\partial}{\partial b}\mathcal{L}$. 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 as to compute the gradients to current batch and update the parameters according to the gradients.
  - `LinearRegressor.predict` method 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 $\| \underbrace{\nabla_{\underline{\boldsymbol{w}}} \mathcal{L}(\underline{\boldsymbol{w}}, b)}_{\text{analytic}} - \underbrace{\boldsymbol{u}_{\delta_w}(\underline{\boldsymbol{w}}, b)}_{\text{numeric}} \|_2$, $\| \underbrace{\frac{\partial}{\partial b} \mathcal{L}(\underline{\boldsymbol{w}}, b)}_{\text{analytic}} - \underbrace{\boldsymbol{u}_{\delta_b}(\underline{\boldsymbol{w}}, b)}_{\text{numeric}} \|_2$ as a function of δ_w, δ_b (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 the **normalized** dataset and `VirusScore` as our target, generate a plot that compares the numerical gradients to the analytic gradients.
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 `VirusScore` subset training labels.

In your report: Attach the resulting plots. Briefly discuss and justify the demonstrated behavior.

Task: Copy the function given in *test_lr.py* into your notebook / project.
Read and understand the function but do not edit it.

(Q5) In this question, we 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 **normalized** validation subset.
`y_val` should hold the `VirusScore` 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).

Attach the plots to your report, briefly discuss the results and justify the demonstrated behaviors.

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

Section 3: Evaluation and Baseline

Our general goal is to minimize the generalization MSE, that is $E_{(x,y)\sim D}[(h(x)-y)^2]$.

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 will again 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, use `sklearn` to perform cross-validation on the training set to evaluate the performance of the models. As explained earlier, the metric we use for regression is the MSE (when calling `cross_validate`, set `scoring='neg_mean_squared_error'`).

We now train a simple DummyRegressor that always predicts the average `VirusScore` 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.

(Q6) Create a DummyRegressor. Evaluate its performance using `cross-validation`:

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2		

Task: Retrain the dummy regressor on the entire training set and save it for future use (Sec 7).

Section 4: Ridge linear regression

In this section and in the following sections we will tune hyperparameters. Specifically, 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 **original** training set for the remainder of the experiments.

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.
iii. Plotting the 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 baseline dummy regressor.
iv. Reporting the value that yields the optimal validation error and its respective error.

We will now learn to predict `VirusScore` using the sklearn.linear_model.Ridge regressor. Make sure your models are non-homogeneous (`fit_intercept=True`).

(Q7) Tune the regularization strength of the regressor. Remember to attach the required plot and specify the optimal strength with its validation error.

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

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	3	filled	filled
Ridge linear	4		

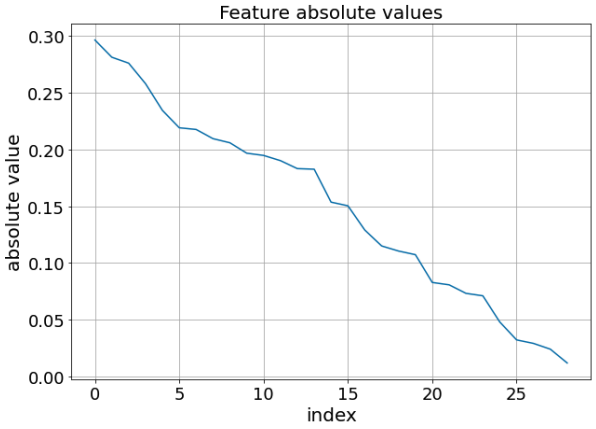
Task: Using the best performing hyperparameter, **retrain** a regressor on the entire training set.

(Q9) Specify the 5 features that have 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 next illustration (the values are made up).



Task: Save this regressor for future use (Section 7).

Section 5: Lasso linear regression

In this section, follow the same instructions as in the previous section, with one difference – instead of the Ridge regressor, use the sklearn.linear_model.Lasso regressor.

(Q11) Tune the regularization strength of the regressor.
Remember to attach the required plot and the optimal strength with its validation error.

(Q12) Is the graph different from the graph tunning λ in the ridge regressor? Explain.

(Q13) Fill in the cross-validated errors of the regressor yielded by the optimal hyperparameter.

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	3	filled	filled
Ridge linear	4	filled	filled
Lasso linear	5		

Task: Using the best performing hyperparameter, **retrain** a regressor on the entire training set.

(Q14) Specify the 5 features that have the 5 largest coefficients (in absolute value) in the resulting regressor, from the largest to the smallest (among these 5).

(Q15) Again, sort and plot the absolute value of the learned coefficients. The x-axis should be the index of the parameter from largest to smallest.
Compare the coefficients obtained here to the ones of the Ridge regressor from (Q10).

Task: Save this regressor for future use (Section 7).

Section 6: Polynomial fitting

We will now try to improve our linear models by adding quadratic features.

Task: Create copies of your train and test sets. Add 2nd-degree polynomial features (`degree=2, include_bias=False`).

We are going to run our regression models using the described feature mapping. We denote the class of all possible (2nd-degree) hypotheses by $\mathcal{H}_{\text{poly}}$.

(Q16) Explain how (and why) we can expect the training and validation errors (each) to change when using such a mapping.

One student suggested using the answer to (Q1) to create a hierarchical model (i.e., a multilevel model) rather than using a polynomial mapping. In the hierarchical model, samples are first divided into two subsets, depending on the condition from (Q1). Then, a different linear regressor is used for each subset (two different linear regressors).

That is, we can learn separate $\underline{\boldsymbol{w}}_1, \underline{\boldsymbol{w}}_2, b_1, b_2$ and build a "conditioned" hypothesis:

$$h_{\text{multi}}(\underline{\boldsymbol{x}}) = \begin{cases} \underline{\boldsymbol{w}}_1^{\top}\underline{\boldsymbol{x}} + b_1, & x \text{ answers the condition in Q1} \\ \underline{\boldsymbol{w}}_2^{\top}\underline{\boldsymbol{x}} + b_2, & \text{otherwise} \end{cases}$$

The student claimed that since we added the indicator-feature in (Q1), a hierarchical model is a special case of the 2nd-degree polynomial mapping (in the sense that an equivalent model can be learned by fitting 2nd-degree polynomials).

(Q17) Explain why the student is correct. Show mathematically that $h_{\text{multi}}(\underline{\boldsymbol{x}}) \in \mathcal{H}_{\text{poly}}$.
When learning both a hierarchical model and a 2nd-degree polynomial model, which model will yield a better train and validation errors? Explain in detail.

(Q18) Tune the regularization strength of a Ridge regressor (with the polynomial mapping). Remember to attach required plots, optimal strengths, optimal validation errors.

(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	
⋮	⋮	⋮	⋮
Ridge polynomial	6		

Task: Train the polynomial regressor on the entire training set and save it for future use (Sec 7).

Section 7: 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 those 3 models, as well as the Dummy baseline.

Remember: do not cross-validate here. Train on the entire training set and evaluate on the test set.

(Q20) Complete the entire table.

Model	Section	Train MSE	Valid MSE	Test MSE
		Cross validated		Retrained
Dummy	3	filled	filled	
Ridge linear	4	filled	filled	
Lasso linear	5	filled	filled	
Ridge polynomial	6	filled	filled	

Which model performed best on the test set?

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

Section 8: Custom models competition (bonus)

This is the section where you are given the freedom to choose your own models. Train a regression model(s) (any model) from the sklearn library only. You can train ensembles of models, ElasticNet (combines L1 and L2 regularization), gradient boosting trees, multilayer perceptrons (neural networks), and more.

You should remember that better features will give you better results. You can use any feature mapping you like and/or employ any insight you have on the features and their interactions with each other or with the target variable.

You might also benefit from brief additional data exploration steps with respect to the updated target variable `VirusScore` (see for instance seaborn.regplot).

Your goal

Find a model that achieves the lowest generalization MSE (think how to estimate it in the best way). You should train using the labeled dataset you worked with so far, and finally predict labels for the unlabeled dataset in the next section.

Competition and Grading (bonus part)

In the next section, you will predict labels for an **unlabeled** dataset and submit them. We will compute the MSE of your predictions using the ground truth labels.

Finally, we will rank the submissions according to their MSEs. The team with the lowest MSE will be awarded with 2 bonus points to their final grade in the course. The teams with the 2nd-4th lowest MSEs, will be awarded with a single bonus point to their final grade.

Bonus awards will only be granted to teams that will answer the following question:

(Q21) (Bonus) Explain which model you used in this section. Describe any special feature mappings or any additional data transformations you used (and why you chose them). Describe the selection process you used. This question should take 1-2 pages.

Section 9: Submitted model predictions

For each model you implemented, you should submit a single CSV file containing your predictions on the unlabeled dataset `virus_unlabeled.csv`.

Notice that this unseen dataset requires the same preparation as the rest of your data.

Before performing any predictions, train each of your models on the **entire labeled dataset at your disposal**, including the labeled test set.

For each relevant section, submit a file named **`pred_<sect_num>.csv`**.

Each file should consist of an "`patient_id`" column and a "`VirusScore`" column.

Every row should have the original ID of the predicted input and the prediction of the best regressor you found in the appropriate section.

Example for a prediction file (actual file should have 1001 rows including the headers):

	A	B
1	patient_id	VirusScore
2	8486	0.15
3	4238	0.43
4	9919	0.29
5	6142	0.35
6	671	0.23

Important: prediction files not complying with this template may result in a point reduction.

In your submitted *zip* file, include the following files:

- Non-comparative sections (10pts):

  Here we simply verify that your predictions "make sense" and MSEs are low enough.

  - **`pred_4.csv`** Best ridge linear model's predictions.
  - **`pred_5.csv`** Best LASSO linear model's predictions.
  - **`pred_6.csv`** Best polynomial model's predictions.

- Comparative bonus section:

  Here you compete with the other submissions, see previous section.

  - **`pred_8.csv`** Best custom models' predictions.

NOTE: do not submit predictions with a false model, e.g., `pred_4.csv` with a polynomial model's predictions. We know the limitations of the models.