CITS5508 Machine Learning
Semester 1, 2024

Assignment 1 - general feedback

This is a general feedback regarding Assignment 1. For the majority of the tasks, it is described what was expected or the correct approach to earn full marks, including code examples as a guide when necessary. Please note that there are different ways to code a specific function/problem.

Overall:

- Some students did not submit both files (the code and the report);
- Some students used any other file format other than PDF for the report and IPYNB for the code.
- Many students were not careful in the presentation of the results, e.g. not formatting tables, presenting the confusion matrix without showing the class names, presenting diagrams/plots without proper titles, not adding axis labels, etc.
- Data normalisation was not necessary, as all features are pixels in the same range. However, you may have presented the features in the range 0 (black) and 1 (white) and/or a justification that is coherent, e.g. for the L2 regularisation, changing the scale to 0-1 and/or normalising the data may help faster/better convergence of the algorithm.
- Many students did not include the random state seeds in all functions that have a "random_state" argument.
- Many students used the `scikit-learn` libraries function without carefully understanding the implementation details and the several arguments each function will need to be set. This has an impact on the results. For instance, the class `SGDClassifier` has an argument `fit_intercept`, which is "True" by default. Then, if in your implementation you are adding the unit feature to represent the bias term, you will need to adapt it before using `SGDClassifier`. This class also has, by default, `learning_rate='optimal'`, so you should understand the impact of this on the results. There are many other examples in the library classes used in the assignment, and the students are expected to make the appropriate adjustments/settings when using the classes.
- After fine-tuning the hyperparameters (using k-fold or training/validation approaches), many students didn't use the entire training set to fit the model and then compute the necessary measures in the test data.

- Some students did data standardisation, which involves adjusting features to have a zero mean and unit standard deviation. However, it is crucial to estimate these parameters (the mean and standard deviation of each feature) solely based on the training set. Applying these estimated parameters directly to the test set ensures consistency and accuracy in the normalisation process. Relying on the test set to estimate these parameters can lead to biased results (you are using information from the test set to normalise the features). So, if you are using `StandardScaler()`, you should use `fit_transform` on the training set and only `transform` on the test set.

D1

Correct answers presented a well-formatted table with the correct number of instances for the three datasets (training, test, and total). Of the 60,000 training images, there was supposed to be a balanced dataset of 6000 images per category. However, there is a coincidental mismatch between groups [5,7]. Students would work with the 5994 examples per group, but keep in mind that it may have mislabelled examples in the data.

D2

Correct answers contain:

- Comment on the balanced/imbalanced aspect.
- Well-formatted bar plot with correct numbers.

D3

Correct answers contain:

- Correct example IDs and inclusion of six images from each class.
- Sandals have label 1 and sneakers have label 0.
- Images are well-formatted (with examples ids and labels).

D4

Correct answers contain:

- Correct use of the bias term and unit feature.
- Your own implementation.
- Correct loss and gradient functions.
- Correct number of iterations, that is, 10,000.

- Correct initialisation of the parameter vector for each η.
- Well-formatted plots.
- Coherent plots that support the explanation.
- Correct split and setting of the random generator seed.

Your loss function should look something like:

```
def loss(beta, x, y, n):
    linear = x @ beta
    p = 1/(1+np.exp(-linear))

    # Handling with log(0)
    p[p == 0] = 0.0000001
    p[p == 1] = 1 - 0.0000001

    aux = -(sum(np.log(p[y == 1]))+np.sum(np.log(1-p[y == 0])))/n

    return aux
```

Your gradient function should look something like:

```
def grad(beta, x, y, n):
    linear = x @ beta
    p = 1/(1+np.exp(-linear))

    return ((x.T @ (p - y))/n)
```

D5

Correct answers contain:

- Correct computation of the loss in the training and validation sets.
- The value for η is defined as 0.00001.
- The bias term was properly introduced in the validation set.
- Use of the same loss function as before and correct calculation of the misclassification ratio.
- Use of your own implementation (no package functions).
- Use of the model (with threshold = 0.5) to get the fraction of misclassifications (e.g. using something similar to the `apply_logistic` function below.)

- Correct update of the parameters after each iteration.

```
def apply_logistic(beta, x, threshold):
    linear = x @ beta
    p = 1/(1+np.exp(-linear))

    label = np.zeros((len(p),1))

    label[p < threshold] = 0
    label[p >= threshold] = 1

    return label
```

D6

Reasonable comments about training and validation loss, and training and validation misclassification.

D7

Correct answers contain:

- Correct computing of the loss function in the validation set (the regularisation term wasn't included).
- Correct derivatives with the inclusion of the regularisation parameter.
- Correct loss function with the inclusion of the regularisation parameter.
- Correct α vector.
- Use of your own implementation.
- Correct value for η.
- Correct use of the model (with threshold = 0.5) to get the fraction of misclassifications (e.g. using something similar to the `apply_logistic` function).
- Use of the loss function similarly as before and correct calculation of the misclassification ratio.
- Well-formatted plots.

```
def loss_reg(beta, x, y, n, C):
    linear = x @ beta
    p = 1/(1+np.exp(-linear))
    theta = beta.copy()
    theta[0] = 0

    # Handling with log(0)
    p[p == 0] = 0.0000001
    p[p == 1] = 1 - 0.0000001

    aux = -(sum(np.log(p[y == 1]))+np.sum(np.log(1-p[y == 0])))/n + (C*np.sum(theta**2))/n

    return aux


def grad_reg(beta, x, y, n, C):
    linear = x @ beta
    p = 1/(1+np.exp(-linear))
    theta = beta.copy()
    theta[0] = 0

    return (((x.T @ (p - y)) + (2*C*theta))/n)
```

D8

Correct answers would contain:

- Correct plot of the cost function.
- Correct plot of the misclassification fraction.
- Correct use of C, that is, considering the way `LogisticRegressionCV` uses it (as the inverse of the regularisation parameter).

Unfortunately, python class `sklearn.linear_model.LogisticRegressionCV` does not provide a simple/direct way to access the necessary values to answer task D8. In this case, all students will receive 4 marks for this task.

D9

Correct answers would contain:

- An answer for which regularisation parameter to use and the reason for that.
- Comment on the impact of the cross-validation.
- Comparison with using 10-fold cross-validation and one validation set.

Given there is no simple/direct way to compute D8, all students earned three marks for task D9.

D10

Correct answers contain:

- Correct computing the cost function in the training and validation sets (for instance, not including the regularisation in the val)
- Correct values for the regularisation parameter.
- Correct use of the `SGDClassifier`.
- Use of the training set without the fixed validation set.
- Correct use of bias in the `SGDClassifier class`.
- Correct scoring argument for the Grid-Search (`neg_log_loss`).
- Correct use of the parameters to compute the loss and the misclassification.
- Correct use of the optimal α to calculate the loss and misclassification.
- Properly formatted output.

D11

Reasonable interpretation.
Comparison with task D8 is not possible. See comment on D8. All students earn appropriate marks for this part.

D12

Correct answers contain:

- Correct use of the probability of the positive class, e.g. `precision_recall_curve(y, y_pred_prob[:,1])`
- Use of the optimal model as obtained in the previous task.
- Use of the fixed validation set and using `predict_proba`.
- Correct use of the prediction vs. recall curve.
- Explanation of how the measure behaves and which threshold to use.
- Well-formatted plots.

D13

Correct answers contain:

- Answers to the question and comment on the comparison.
- Correct use of the model, similar to using `apply_logistic` and varying the value of the threshold.
- Correct use of the fixed validation set.
- Computation of the performance metrics on the fixed validation set.

D14

Correct answers contain:

- LR2: the value selected for the regularisation parameter and why.
- Correct use of this value in your function.
- Fitting the models on the entire training set.
- LR3: Correct calling the `SGDClassifier` class, e.g.:

```
SGDClassifier(loss='log_loss', random_state=5508, alpha=best_alpha)
```

- Correct use of the parameters when using the function to make predictions on the test set, e.g.

```
... = np.row_stack((sgd_clf.intercept_,
                  np.reshape(sgd_clf.coef_[0], (784,1))))
```

- Use of 0.5 threshold on all but the last model, e.g.

```
... = apply_logistic(param_clf, xtest, 0.5) #L3
... = apply_logistic(param_clf, xtest, best_threshold) #L4
```

- Well-formatted outputs.

D15

Reasonable comments about the results and comparison of the generalisation capacity of the models.

D16

Correct answers contain:

- Plotting the images using the test set and using the L4 model.
- Proper formatting of the plots, including the labels.
- Correct calculation of false positives and false negatives, e.g.

```
fp_indices = np.where((ytest == 0) & (y_pred == 1))[0][:5]
fn_indices = np.where((ytest == 1) & (y_pred == 0))[0][:5]
```

D17

Correct answers contain:

- Discuss about the misclassification.
- Notice the data labelling problem (see comment on D1). A sneaker was correctly predicted as a sneaker but is mislabelled in the test data as sandals, leading to a false negative.

D18

Correct answers contain:

- Use of the correct model (L4).
- The heatmap is readable, well formatted, and the values associated with each pixel are presented.
- The weight's magnitude was taken into account.

D19

Comments may include a discussion about pixels that have negative/positive and high-intensity values that correspond to the negative (sneakers) and positive (sandals) classes.

D20

Correct answers contain:

- Using the correct training set and validation set.
- Correct curves for training and validation.
- Well-formatted plots.

D21

Correct answers contain:

- Answer about which value of k to use.
- Reasonable comment on the behaviour of the training and validation curves as k increases.
- Comparison with the model in D5.

D22

Correct answers contain:

- Correct use of k-NN with the entire training set and the best value of k selected.
- Correct calculation of the confusion matrix on the test set.
- Properly formatting the outputs, e.g. including the labels in the confusion matrix.

D23

Discussions may include a reflection about the precision and recall, an analysis of the performance concerning the complexity of the model or convergence issues, the nature of the data (high-dimensional), how reasonable the selection of the range for α values was, the fact that k-NN had the worse recall but almost 1.0 precision, etc.

D24

Possible answers may have: adding the number of non-constant pixels as a new feature, adding a feature that accounts for the shape/contour of the pixels, performing feature selection as many pixels are constant or correlated, using a Kernel trick, creating other types of new features, extend the range on the grid search for the regularisation parameter, etc.