CS 441

HW 2: PCA and Linear Models

Due Date: Sep 29, 2025 11:59pm

In this assignment we will explore linear projection, classification, and regression methods, again on digit recognition and temperature prediction.

The aims of this homework are:

1. Become familiar with PCA, Linear Logistic Regression, SVM, and Linear Regression methods.
2. Gain practice using visualizations and analysis to better understand the models and their effectiveness
3. Gain experience in classification and regression applications

Read this document and the report template and tips and tricks before beginning to code.

- Report template
- Tips and Tricks
- Starter code and data

1. PCA on MNIST [30 points]

Compute the principal components using sklearn.decomposition.PCA over the *full training set*.

a. *Display the first 10 principal components* using the same tool that is used to display centroids (`display_mnist`).
b. *Scatterplot the first two dimensions of PCA-transformed* `x_train[:500]`. Show a different color for each digit label.
c. *Plot cumulative explained variance (*`explained_variance_ratio_`*)* of all components.
d. Select the smallest number M of principal components that explains at least 90% of variance. Then, compress the training data and the test data using the first M principal components, and compute the total time and test error for 1-NN using the brute force Faiss method. Fit PCA on the training data only, then use the same components to transform both training and test data. *Compare time and error for 1-NN using original vs compressed features.*

2. MNIST Classification with Linear Models [45 points]

We revisit classification using linear logistic regression (LLR) and SVM. For logistic regression, use `sklearn.linear_model.LogisticRegression` with default parameters (unless stated otherwise), except `max_iter=500`. For SVM, use `sklearn.svm.LinearSVC` with default parameters, except `max_iter=500`. Do not use `svm.SVC` since that is a non-linear SVM by default.

Note that it can take several minutes for each method to train on the full training set. I typically work on the next thing while a model is training. In practice, training ML algorithms can take days. With `max_iter=500`, you may get a warning that it hasn't converged, and that's ok. We're using a smaller number of iterations than ideal to save time.

a. Train and test Linear Logistic Regression (LLR) and SVM

For varying size training sets N in [100, 1000, 10000, 60000], *compare and tabulate classification error for linear logistic regression and linear SVM*, using `x_train[:N]` and `x_test`.

b. Error visualization

For each of LLR and SVM (full training set only), for each label (0 to 9), *display the sample with the highest score for the correct label.* E.g., out of all samples with a true label of 0, select the one that has the highest score for label 0. Display the 10 samples in a row for each method, showing the easy cases for each model. Then, for each label, *display the sample with the lowest score for the correct label*. Again, display the 10 samples in a row, showing the difficult cases.

c. Parameter Selection

For linear SVM, use validation experiments (testing on the validation set) to estimate the best regularization strength `C`. Then evaluate a model trained with that `C` on the test set. For the sake of speed, in this section, use `x_train[:1000]` as the training set for all experiments, with `max_iter=1000`. For the validation set, use `x_train[50000:]`.

In selection, try to get the best result on the validation set within 0.1%. To select `C`, start with the default and increase/decrease by a factor of 2 until results either level out or start to get worse. Then, you can try half way between the two best values. E.g. try `C` in [0.25, 0.5, 1, 2, 4]. Suppose 0.5 and 1 give the best results. Then, try 0.75. Once the difference between the two best results is less than 0.1%, you can stop. You can also search outside of this range if 0.25 or 4 gives the best result. *Plot the validation error for each C value tested on a semilogx plot, and record the best in a table.*

3. Temperature Regression [25 points]

Forecasting, predicting a future value based on past values, is a common machine learning problem, which can apply to predicting prices, production, temperature, and many other things. The Temperature Regression problem is derived from a Kaggle dataset. The task is to predict the average temperature in Cleveland for the next day given the average temperature of major US cities in the preceding five days. The dataset has three splits: train (used to fit parameters), val (used for evaluation to select parameters), and test (used for final evaluation). For each split, you have "x", the features used for prediction, and "y", the values you should try to predict.

Each feature column in “x” corresponds to a temperature for some city for some previous day. You can find out which day or city using `feature_to_day` and `feature_to_city`.

We previously investigated using K nearest neighbor for prediction. Now, we will investigate using linear regression for both prediction and feature selection.

a. Train and Test Linear Regression (LR)

Report RMSE results for L2-regularized linear regression (`sklearn.linear_model.Ridge`) using default regularization parameters on the temperature regression dataset, using original and normalized features (based on feature number 361, as defined in HW1). When normalizing, remember to normalize both the training and the test sets.

b. Most important features

Use original features for this part. Identify the most important features by: (1) fitting a L1 Linear Regression model (`sklearn.linear_model.Lasso`) with default parameters; and (2) selecting the features that have coefficient magnitudes greater than 0.001 (`model.coef_` is the linear coefficient vector; -0.02 and 0.02 both have magnitude greater than 0.001). For the top 10 features, report their indices and corresponding city and day. Also report the test RMSE error rates of L2 (ridge) linear regression when trained *using only the top 10 selected features*.

Reflections (not reported/graded): Think about the cities and days of the best features – does it tell you anything about how the temperature tends to move across the country? How does your linear regression compare in performance in interpretability to K-NN from HW1?

4. Stretch Goals

a. **PR and ROC curves** [10 points]

Often, we want to evaluate performance in a threshold-invariant way, or to see how classification varies with the confidence threshold. On the MNIST data, for LLR with default parameters, create precision-recall and ROC plots for predicting whether a digit is a “0” vs. not “0”. You can use any libraries to do this. Also report AP(Average Precision) and Area Under the Curve (AUC).

b. **Visualize weights** [10 points]

For each digit (0–9), display the linear classifier weights as an image. Put all the images in a row. Do this for the LLR weights using L1, L2, and elastic regularization (parameters can be default or of your choice) and for SVM weights. It’s OK to use a smaller training size for this, e.g. first 1000 samples – the main purpose is to visualize how regularization affects the weights.

c. **Other embeddings** [15 points]

Extending Part 1b, plot 2D points of `x_train[:500]` using at least two of t-SNE, MDS, UMAP, and Linear Discriminant Analysis, and display these plots alongside the PCA scatterplot. You may fit/train on 500 points as well.

d. **One city is all you need?** [15 points]

Suppose you are only allowed to know the temperature for one city from the previous five days. You can use any feature normalization or regression model. Which city is best, and what RMSE do you get in testing? Remember that for model selection (including choosing a city), you should use a validation set, only testing once you have made a selection. Explain your process.

e. **SVM with RBF Kernel** [10 points]

Compare results across training sizes for linear SVM (`LinearSVC`) and SVM with an RBF kernel, which is the default kernel when using `SVC`. For linear, you can copy-paste your results from 2a.

Submission Instructions

Append two files: (1) completed report template; (2) a pdf version of your Jupyter notebook. Be sure to include your name and acknowledgments. **The report is your primary deliverable**. The notebook pdf is included for verification/clarification by the grader. To create PDF of notebook: Use "jupyter nbconvert" -- see starter code. To combine PDFs: use https://combinepdf.com/ or another free merge tool. *The first pages should correspond to the report, followed by the code.*

Submit the combined file to Gradescope HW2. **In Gradescope, select pages in your template *and* notebook that correspond to each subproblem**. Failure to correctly identify pages will result in a penalty.