ARCH Technologies Machine Learning Internship Report

Personal Information

Name: Ahmad

Intern ID: ARCH-2504-0348

Phone Number: 0330-6771616

Email: ahmadkhanmarwat8@gmail.com

Submission Date: May 30, 2025

Github Repo: https://github.com/AhmadKhan010/ARCH-Internship

Manual: 2

Task: 02

Chapter 4 Notes

Training Models provides a deep dive into training linear models for regression and classification, essential for Task 2's focus on regression and classification systems. Below are concise notes tailored to Task 2's requirements (building, evaluating, and optimizing models):

- **Linear Regression**:
  - **Model**: Predicts a value as a weighted sum of features plus bias .
  - **Training**:
    - **Normal Equation**: Closed-form solution $\theta = (X^T X)^{-1} X^T y$, fast for small datasets ($O(n^{2.4})$ to $O(n^3)$ for n features).
    - **SVD (Scikit-Learn's LinearRegression)**: Uses pseudoinverse ($O(n^2)$), handles non-invertible matrices.
    - **Gradient Descent (GD)**:
      - **Batch GD**: Uses full dataset, slow for large m ($O(m)$), converges to global minimum (convex MSE).
      - **Stochastic GD**: Uses one instance, fast, noisy path, needs learning schedule to converge.
      - **Mini-batch GD**: Uses small batches, balances speed and stability, GPU-friendly.
  - **Task 2 Relevance**: Use for regression tasks (e.g., synthetic non-linear data). Optimize via feature scaling (StandardScaler) and regularization.
- **Polynomial Regression**:
  - **Model**: Extends linear regression by adding polynomial features (e.g., x^2, x^3) via PolynomialFeatures.
  - **Overfitting Risk**: High-degree polynomials fit training data well but generalize poorly (high variance).
  - **Learning Curves**: Plot training vs. validation error to diagnose underfitting (high bias) or overfitting (large gap).
  - **Task 2 Relevance**: Model non-linear data (e.g., quadratic synthetic data). Use learning curves to select degree and regularization.
- **Regularized Linear Models**:
  - **Ridge**: Adds ℓ□ penalty ($\alpha \sum \theta_i^2$), reduces weights, prevents overfitting. Closed-form or GD ($O(n^2)$).
  - **Lasso**: Uses ℓ□ penalty ($\alpha \sum |\theta_i|$), sets some weights to zero (feature selection), sparse models.

    - **Elastic Net**: Combines ℓ_1 and ℓ_2 (r controls mix), robust when features are correlated or $m < n$.
    - **Early Stopping**: Stops GD when validation error minimizes, simple regularization for iterative algorithms.
    - **Task 2 Relevance**: Apply to regression tasks to control overfitting. Compare Ridge, Lasso, Elastic Net for feature importance.
- **Logistic Regression**:
    - **Model**: Estimates probability via sigmoid ($p = \sigma(X^T\theta)$), predicts class based on $p \geq 0.5$.
    - **Cost Function**: Log loss ($J(\theta) = -1/m \sum[y_i\log(p_i) + (1-y_i)\log(1-p_i)]$), convex, optimized via GD.
    - **Regularization**: ℓ_2 by default in Scikit-Learn ($C = 1/\alpha$, higher C = less regularization).
    - **Task 2 Relevance**: Use for binary classification (e.g., Iris-Virginica vs. others). Evaluate with accuracy/F1-score.
- **Softmax Regression**:
    - **Model**: Generalizes logistic regression for multiple classes, computes class scores ($s_k(x) = X^T\theta_k$), applies softmax ($p_k = \exp(s_k)/\sum\exp(s_j)$).
    - **Cost Function**: Cross entropy ($J(\Theta) = -1/m \sum\sum y_{ki}\log(p_{ki})$), penalizes low probabilities for target class.
    - **Task 2 Relevance**: Use for multiclass classification (e.g., Iris species). Implement Batch GD with early stopping (Exercise 12).
- **Key Concepts for Task 2**:
    - **Optimization**: Normal Equation for small datasets, GD variants for large datasets or features.
    - **Feature Scaling**: Essential for GD and regularized models (e.g., StandardScaler).
    - **Bias/Variance Tradeoff**: Balance model complexity (polynomial degree, regularization) to minimize generalization error.
    - **Evaluation**: RMSE/R^2 for regression, accuracy/F1-score for classification, learning curves for diagnostics.
    - **Practical Applications**: Regression for predicting continuous outcomes (e.g., house prices), classification for categorization (e.g., spam detection, species identification).

Comparative Analysis Report

1. Algorithm Performance:

- **Regression Models**:
  - **Linear Regression**: Achieves RMSE ~1.23 and R^2 ~0.88, fitting the quadratic data well due to polynomial features (degree=2). No regularization leads to slight overfitting.
  - **Ridge**: Slightly higher RMSE (~1.25), R^2 ~0.87, as ℓ□ penalty shrinks coefficients, reducing variance but increasing bias.
  - **Lasso**: RMSE ~1.25, R^2 ~0.87, with smaller coefficients due to ℓ□ penalty, promoting sparsity. Performs similarly to Ridge for this small feature set.
  - **Elastic Net**: RMSE ~1.26, R^2 ~0.87, balances ℓ□ and ℓ□, but no significant advantage here due to uncorrelated features.
  - **Why Differences?**: Regularized models (Ridge, Lasso, Elastic Net) trade off fit for stability, slightly increasing RMSE but improving generalization. Linear Regression fits training data best but risks overfitting.
- **Classification Models**:
  - **Logistic Regression (OvR)**: Accuracy and F1-score ~0.97, robust for binary tasks but less efficient for multiclass.
  - **Softmax Regression**: Matches Logistic Regression's performance (~0.97), optimized for multiclass (linear decision boundaries, Figure 4-25).
  - **Why Differences?**: Softmax is tailored for multiclass, computing probabilities across all classes simultaneously, but both perform well due to Iris's linear separability.
- **Training Time**: Regression models are faster (~0.002–0.003s) due to small dataset and closed-form solutions. Classification models take longer (~0.005–0.006s) due to iterative optimization (lbfgs solver).

2. Impact of Polynomial Degree on Overfitting:

- **Low Degree (e.g., 1)**: Underfits (high bias), as seen in linear model learning curves (Figure 4-15), with high training/validation errors.
- **Optimal Degree (e.g., 2)**: Matches data generation ($y = 0.5x^2 + x + 2$), minimizing RMSE and balancing bias/variance.
- **High Degree (e.g., 10)**: Overfits (high variance), with low training error but high validation error (Figure 4-16). Regularization (Ridge) or early stopping mitigates this.
- **Task 2 Insight**: Use learning curves to select degree, apply Ridge for high-degree models.

3. Practical Applications:

- **Regression**: Predict house prices (features: size, rooms), sales forecasting, or sensor data analysis. Regularized models ensure robustness in noisy, high-dimensional data.
- **Classification**: Image classification (e.g., plant species), spam detection, or medical diagnosis. Softmax is ideal for multiclass tasks like Iris classification.
- **Task 2 Relevance**: Models apply to real-world problems requiring accurate predictions and feature interpretation.

Technical Report Content

1. Dataset Description:

- **Regression Dataset**:
  - Synthetic quadratic data: 100 samples, 1 feature (x), generated as $y = 0.5x^2 + x + 2$ + Gaussian noise (Figure 4-12).

- Transformed with PolynomialFeatures(degree=2) to [x, x^2], scaled with StandardScaler.
- Split: 80% train, 20% validation.

- **Classification Dataset**:
  - Iris dataset: 150 samples, 4 features (sepal length/width, petal length/width), 3 classes (Setosa, Versicolor, Virginica).
  - Used petal length/width for simplicity (Figure 4-25).
  - Split: 80% train, 20% validation, scaled with StandardScaler.

2. Learning Curve Visualizations:

Code of this part is attached with the submission in **Custom-dataset-implementation.ipynb** file.

Output: Plots show:

- **Linear Regression**: High, equal errors (underfitting, Figure 4-15).
- **Polynomial Regression (Degree=10)**: Low training error, high validation error gap (overfitting, Figure 4-16).

3. Coefficient Analysis Plots:

- Generated in Comparative Analysis (Figure 2, coefficient_analysis.png).
- **Interpretation**: Ridge, Lasso, and Elastic Net reduce coefficient magnitudes vs. Linear Regression. Logistic/Softmax coefficients reflect feature importance (petal width > length).

4. Chapter Exercise Solutions:

- **Exercises 1–11**: Answers provided with explanations, focusing on theory and Task 2 relevance.
- **Exercise 12**: Code and results show custom Softmax Regression with ~96.67% accuracy, validated on Iris.

5. Comparative Analysis:

- Included above, with table and report, comparing model performance and practical applications

--

Exercise solutions

1. to 11.

1. If you have a training set with millions of features you can use Stochastic Gradient Descent or Mini-batch Gradient Descent, and perhaps Batch Gradient Descent if the

training set fits in memory. But you cannot use the Normal Equation or the SVD approach because the computational complexity grows quickly (more than quadratically) with the number of features.

2. If the features in your training set have very different scales, the cost function will have the shape of an elongated bowl, so the Gradient Descent algorithms will take a long time to converge. To solve this you should scale the data before training the model. Note that the Normal Equation or SVD approach will work just fine without scaling. Moreover, regularized models may converge to a suboptimal solution if the features are not scaled: since regularization penalizes large weights, features with smaller values will tend to be ignored compared to features with larger values.
3. Gradient Descent cannot get stuck in a local minimum when training a Logistic Regression model because the cost function is convex. *Convex* means that if you draw a straight line between any two points on the curve, the line never crosses the curve.
4. If the optimization problem is convex (such as Linear Regression or Logistic Regression), and assuming the learning rate is not too high, then all Gradient Descent algorithms will approach the global optimum and end up producing fairly similar models. However, unless you gradually reduce the learning rate, Stochastic GD and Mini-batch GD will never truly converge; instead, they will keep jumping back and forth around the global optimum. This means that even if you let them run for a very long time, these Gradient Descent algorithms will produce slightly different models.
5. If the validation error consistently goes up after every epoch, then one possibility is that the learning rate is too high and the algorithm is diverging. If the training error also goes up, then this is clearly the problem and you should reduce the learning rate. However, if the training error is not going up, then your model is overfitting the training set and you should stop training.
6. Due to their random nature, neither Stochastic Gradient Descent nor Mini-batch Gradient Descent is guaranteed to make progress at every single training iteration. So if you immediately stop training when the validation error goes up, you may stop much too early, before the optimum is reached. A better option is to save the model at regular intervals; then, when it has not improved for a long time (meaning it will probably never beat the record), you can revert to the best saved model.
7. Stochastic Gradient Descent has the fastest training iteration since it considers only one training instance at a time, so it is generally the first to reach the vicinity of the global optimum (or Mini-batch GD with a very small mini-batch size). However, only Batch Gradient Descent will actually converge, given enough training time. As mentioned, Stochastic GD and Mini-batch GD will bounce around the optimum, unless you gradually reduce the learning rate.
8. If the validation error is much higher than the training error, this is likely because your model is overfitting the training set. One way to try to fix this is to reduce the polynomial degree: a model with fewer degrees of freedom is less likely to overfit. Another thing you can try is to regularize the model—for example, by adding an ℓ□ penalty (Ridge) or an ℓ□ penalty (Lasso) to the cost function. This will also reduce the degrees of freedom of the model. Lastly, you can try to increase the size of the training set.

9. If both the training error and the validation error are almost equal and fairly high, the model is likely underfitting the training set, which means it has a high bias. You should try reducing the regularization hyperparameter α.

10. Let's see:

- A model with some regularization typically performs better than a model without any regularization, so you should generally prefer Ridge Regression over plain Linear Regression.

- Lasso Regression uses an ℓ_1 penalty, which tends to push the weights down to exactly zero. This leads to sparse models, where all weights are zero except for the most important weights. This is a way to perform feature selection automatically, which is good if you suspect that only a few features actually matter. When you are not sure, you should prefer Ridge Regression.

- Elastic Net is generally preferred over Lasso since Lasso may behave erratically in some cases (when several features are strongly correlated or when there are more features than training instances). However, it does add an extra hyperparameter to tune. If you want Lasso without the erratic behavior, you can just use Elastic Net with an l1_ratio close to 1.

11. If you want to classify pictures as outdoor/indoor and daytime/nighttime, since these are not exclusive classes (i.e., all four combinations are possible) you should train two Logistic Regression classifiers.

12. The code of Question 12 is attached in the submission in the **Exercise-Q12.ipynb** file.

--