Machine Learning Project Report

Hanming Zhang
School of Computer Science
Shanghai Jiaotong University
Dongchuan Road 800
`chrisming@sjtu.edu.cn`

Abstract

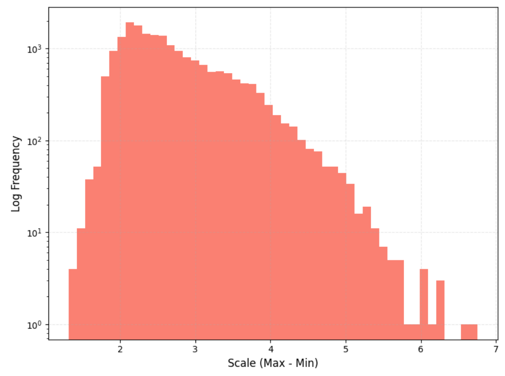
This report investigates a multiclass classification problem using a combination of linear models, kernel-based methods, and ensemble learning techniques. We begin with an exploratory analysis of the data, examining feature sparsity, class imbalance, and the implications of high dimensionality. Several dimensionality reduction methods, including PCA, t-SNE, and ISOMAP, are employed to study the geometric structure of the data and to guide model selection. We then evaluate a range of supervised learning models, including multinomial logistic regression with L1 and L2 regularization, support vector machines, random forests and multiple layer perceptron. Model selection and hyperparameter tuning are conducted using stratified cross validation, with performance assessed primarily via macro F1-score to account for class imbalance. Information-theoretic criteria such as AIC and BIC are further used to analyze the trade-off between model fit and complexity, particularly in the context of Lasso regression. Experimental results demonstrate that appropriate dimensionality reduction and regularization significantly improve generalization performance, with random forest using 1850 estimators achieving the best test accuracy ($\mathbf{90.00}\%$). The report also highlights the non-comparability of regularization scales between L1 and L2 penalties and provides empirical evidence supporting the tendency of BIC to favor simpler models. Overall, the findings illustrate how principled model selection, regularization, and representation learning jointly contribute to robust performance in high-dimensional multiclass classification tasks. The complete kaggle online test results and ranking screenshot are provided in appendixA and B respectively.

1 Data Processing and Dimension Reduction

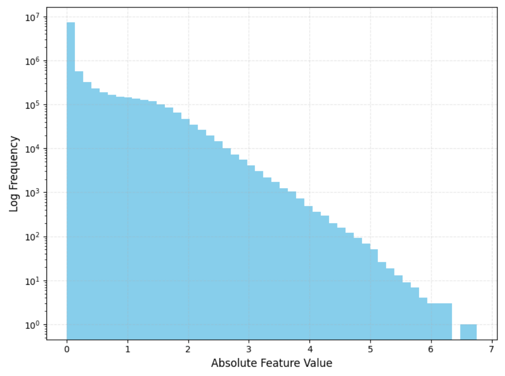
To comprehend the sparse distribution of the input features and class distribution, I have plotted the feature distribution of the training dataset, and adopted dimension reduction techniques–including both linear and non-linear ones–to visualize the distribution of the data.

1.1 Normalization before Dimension Reduction

Since the input data were derived from images using dimension reduction, the scale of the features had changed dramatically. Each feature in a RGB-format image is integer lying in $[0, 255]$, yet after dimension reduction, the scale of each feature can differ significantly. As can be seen in figure[1a], feature scale distribution is not focused enough. Since non-linear dimension reduction techniques such as ISOMAP uses **k-Nearest-Neighbors** as a subprocess, the dimension reduction process is sensitive to the differences of data feature scales ([5][11]). Thus, **normalization** is mandatory to guarantee the effectiveness of the dimension reduction algorithms.



(a) Histogram of feature scales



(b) Histogram of feature magnitudes

Figure 1: (a) Histogram of test data feature scale. X-axis indicates the feature scale $\max - \min$, Y-axis indicates the log frequency of the scale. Most feature scales lie between $[2, 6]$, and inter-class scale distribution is not even. (b) Numeric distribution of raw train data features (without processing). X-axis marks absolute feature value, Y-axis marks the log frequency of feature value. As the feature value increases, log frequency decreases nearly linearly, showing that most features in the train dataset is sparse.

1.2 Sparsity of Features & Sparsity of Samples

By **'sparsity of features'**, we mean that most features in the data are close to zero (cold), only a small number of features are large (hot). Figure 1b shows that the absolute value of most features in the train dataset are small. As is illustrated in section 2.1, decision-tree-based models have natural strength at classifying high-dimension data with sparse features, as **axis-aligned** splits naturally exploits sparsity ([3]).
By **'sparsity of samples'**, we mean that samples that belong to the same class are close to each other, while different classes are distant from each other. In figure[2], the diagonal grids are significantly darker than off-diagonal grids, indicating that in the 512-d feature space, most classes are highly separable, and elements that belong to the same class cluster together. Such sparsely-distributed pattern makes the feature space easy to be separated by linear classifiers.

1.3 Linear and Non-linear Dimension Reduction

Three reduction algorithms were used to provide more insights into train data and cleaner feature space for linear classifiers. For linear dimension reduction, PCA was adopted, for non-linear dimension reduction, ISOMAP ([11]) and t-SNE ([8]) were selected. As is mentioned in section 1.1, normalization on all 512 features was performed before applying non-linear dimension reduction techniques. Only data processed by PCA were used for training models, the reason will be explained in the next paragraph. Below are the 2-d dimension reduction results for the three algorithms. For the sake of simplicity, only the top 20 classes were displayed in the figures.

1.3.1 PCA

PCA serves a dual purpose: providing low-dimensional visualization and performing feature extraction. For the first point, figure 3a shows that most data are highly separable, while the rest overlap dramatically. Thus, it should be easy to achieve relatively high accuracy, but highly unlikely and unworthy to expect further great improvements, as it would be a sign of overfit given the above analysis. For the second point, PCA selects the most indicating directions in the original feature space ([6]), meaning that on these directions, the projected data will have the highest variances. Although L1-regularized logistic regression does provide similar mechanism, no variants of SVM do! As our final results had shown, training SVM on PCA-processed data did offer accuracy improvement–SVM trained on 40-d train data achieved accuracy $\mathbf{89.73\%}$ on the kaggle online test, while the test accuracy SVM trained on the original train data was 89.67%.
There are two tips for applying PCA on the train data. The first one is deciding the number of selected

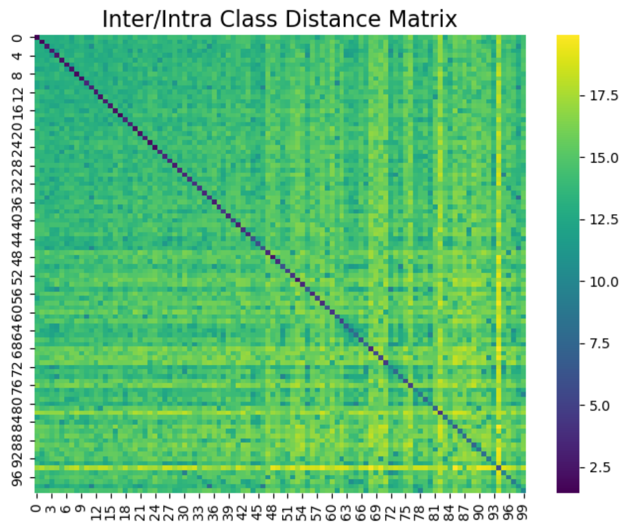
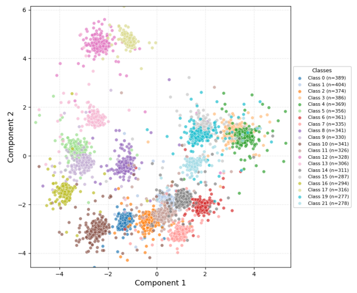


Figure 2: Inter and intra class distance matrix for raw train data (without processing). Inter-class distance between class i and class j is defined as the Euclidean distance between the centroids of the two classes, C_i and C_j. Intra-class distance of class i is defined as the average Euclidean distance of all the class i samples to centroid C_i. The diagonal grid (i, i) corresponds to intra-class distance of class i, off-diagonal grids (i, j), (j, i) correspond to inter-class distance between class i and class j.

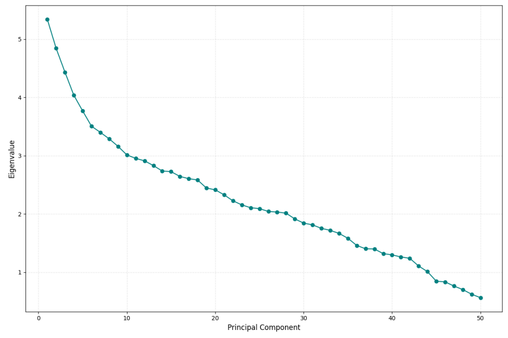
features. Using Kaiser criterion ([7]), which only keeps components whose eigenvalues are no less than 1, the top 44 components were selected. Another more flexible criterion ([6]) suggests choosing the smallest t that satisfies $\frac{\sum_{i=1}^{t}\lambda_i}{\sum_{i=1}^{p}\lambda_i} \geq 1-\epsilon$. For our problem, $p = 512$, and $\epsilon = 16\%$, and the corresponding t is 40. The results showed that the two criterion offered similar prediction accuracy improvements. Another tip is about whether to include test data when applying PCA. Since the test data are not labeled, it surely can't provide any class distribution information. Will test data further improve model prediction accuracy? Yes, but that would be cheating, since the distribution information of the test data will therefore 'leak' into our models ([2]), which shouldn't have any prior knowledge about the test data. Thus, this tip applies to t-SNE and ISOMAP as well.

1.3.2 t-SNE & ISOMAP

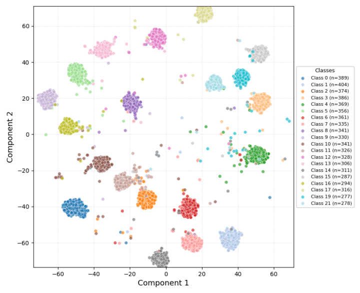
The reason why t-SNE embedding exhibits better cluster structures than ISOMAP lies in their mathematical goals. t-SNE operates within a probabilistic framework, whose primary goal is to preserve local data structure ([8]). In contrast, ISOMAP is a geometric and graph-based technique, whose core objective is to preserve the global geometry of the data manifold ([11]). The use of the **heavy-tailed** t-distribution in the t-SNE effectively alleviates the crowding problem–it forces points of moderate similarity to close together, while pushes dissimilar points apart, thereby creating clear visual margins. ISOMAP, on the other hand, is highly sensitive to **noise and outliers**, which may create short-circuit and corrupt the neighborhood graph. One supervised model that shares the same problem with ISOMAP is the **perceptron**, which also has nice geometric interpretation and poor probabilistic framework.
While the two non-linear dimension reduction algorithms explored in this report only provide class distribution intuition, they should not be used as feature sets in downstream supervised model training. This issue mainly stems from the **non-generalizability** of the two algorithms. In other words, t-SNE and ISOMAP are fundamentally transductive, they compute and embedding for a **fixed**, **static** dataset. To incorporate a new test sample, one must recompute the embedding on the combined dataset, which may drastically alters the existing embedding structure for the train data.



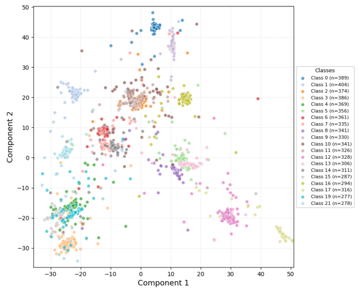
(a) Top 20 classes 2-d distribution after PCA



(b) Top 50 eigenvalues from PCA



(c) Top 20 classes 2-d distribution after t-SNE



(d) Top 20 classes 2-d distribution after ISOMAP

Figure 3: 3a 3c 3d display the 2-d dimension reduction results using PCA, t-SNE, ISOMAP respectively. Figure 3b shows the top 50 eigenvalues from PCA. Both PCA and t-SNE managed to preserve and even magnify the cluster effect of same-class samples, while no clear cluster pattern was observed in the result processed by ISOMAP. Most classes displayed have relatively small variance, making them more separable from other classes. Still there are classes that overlap with others seriously, such as class 15 and 19. The first 50 eigenvalues displayed in figure 3b are informative, since the first 44 are all greater than 1, and the sum of these eigenvalues accounts for 92.19% of total eigenvalues.

2 Adopted Models

Four models were adopted for this classification problem, including two linear models and two non-linear models. Logistic Regression (**LR**), Support Vector Machine (**SVM**) were selected for linear models, Random Forest (**RF**) and Multiple Layer Perceptron (**MLP**) were selected for non-linear ones. Regularization techniques such as L1/L2 penalty were explored. Both LR and SVM had been implemented and compared with sklearn library models. Final results showed that RF using 1850 estimators achieved the best prediction result. SVM was competitive when using PCA-embedding 40-d data and L2 regularization. MLP, however, did not offer extremely high accuracy. A detailed analysis on the outstanding performance of random forest is provided in the following section.

2.1 Decision Tree & Random Forest

Given the sparsity of features, choosing decision-tree-based models is a good choice due to the following 2 specific reasons. 1) **Axis-aligned splits naturally exploit sparsity.** At each node, a decision tree performs splits of the form $x_j \leq t$ on **individual features**. As most coordinates satisfy $x_j \approx 0$, the remaining hot coordinates behave almost like indicator variables. Thus, at the very first few levels, the decision tree can cleanly split the informative non-zero cases. This is a significant advantage over linear models, which may accumulate **weak features** over many dimensions, and distort the clean decision boundaries. 2) **Feature subsampling amplifies sparse signals.** At each split, random forest randomly subsample features, and informative features have higher probability to dominate due to feature's sparsity. Thus, strong features could define early splits and avoid overfit on

the train data. By contrast, linear models consider features jointly, failing to filter out strong signals efficiently. The above analysis explains why random forest managed to achieve the highest accuracy on raw train data.
Yet, random forest still have high variance, so regularization techniques are necessary for model generalization. Three regularization hyper parameters were tuned to select the best model: 1) **n_estimators**: number of decision trees in the random forest 2) **max_features**: number of features subsampled at each split and 3) **class_weight**: weights associated with classes. We will be mainly analyzing the performance gain brought by 1) n_estimators.
Increasing the number of estimators decreases model variance in the most intuitive way. Ideally, when all B decision trees in the model are mutually independent, with each tree's prediction variance $Var(\hat{f}_i(x))$ equals to σ^2, the average prediction variance of all trees $Var(\frac{1}{B}\sum_{i=1}^{B}\hat{f}_i(x))$ would be $\frac{\sigma^2}{B}$. In reality, however, trees in the same model are trained on the same base dataset, and **mutual-independence** no longer holds. We then assume that the average Pearson coefficient between two distinct decision trees in the model is ρ, the forest's variance is now $\frac{1}{B^2}(\sum_{i=1}^{B} Var(\hat{f}_i(x)) + \sum_{i\neq j} Cov(\hat{f}_i(x), \hat{f}_j(x))) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$ ([3]). Thus, increasing the number of estimators B decreases the second term, and the total prediction error thereby approaches to $\rho\sigma^2$. Another guidance offered by the above formula is that, as B grows, to achieve the same model accuracy improvement, the number of new estimators ΔB must increment as well.
Despite the outstanding performance on the original train data, on dimension-reduced dataset, random forest is no longer competitive compared to linear models, as dimension reduction algorithms like PCA fail to preserve feature sparsity. The rotation and shrinkage performed on the original dataset mix the strong indicators and the weak ones. Thus, as previous analysis in this report had mentioned, decision-tree-based models can no longer make axis-aligned splits that separate distinct classes neatly. Non-linear dimension reduction methods (ISOMAP and t-SNE) distort the features even harder. Both methods attempt to cluster samples that are closer to each other in the original space when reducing dimension. In this way, the relative distance relationships were indeed preserved, but there is no guarantee on the sparsity of the new features.

3 Model Assessment and Selection

3.1 Standard Local Evaluation

Since training models on the entire train data and submit the results to kaggle platform is time-consuming, **local tests** were adopted to estimate model accuracy. My standard model evaluation shuffled (using fixed seed to guarantee reproducibility) and split it into train data and test data. The train/test split ratio is set to $2/3$, which in practice reflected model generalizability robustly across all models.

3.2 AIC & BIC

The Akaike Information Criterion (**AIC**) and the Bayesian Information Criterion (**BIC**) are two classic model assessment methods learned in class ([1, 9]). Both criteria are derived from the maximized log-likelihood of the model and introduce an explicit penalty on model complexity.

Let $\hat{L}$ denote the maximized likelihood of the model, $d(\alpha)$ the number of free parameters under hyperparameter α, and N the number of training samples. The AIC is defined as:

$$\text{AIC}(\alpha) = -2\log\hat{L} + 2d(\alpha) \tag{1}$$

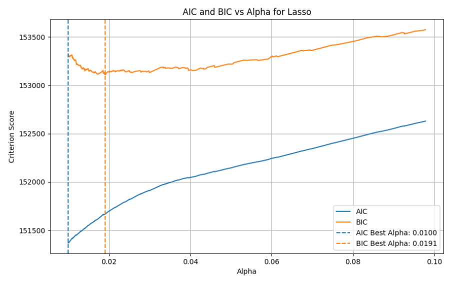
Similarly, the BIC is defined as:

$$\text{BIC}(\alpha) = -2\log\hat{L} + d(\alpha)\log N \tag{2}$$

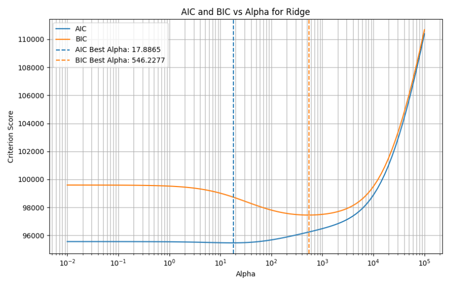
Both criteria favor models with smaller values. The key difference between AIC and BIC lies in the penalty term: while AIC uses a constant penalty $2d(\alpha)$, BIC employs a stronger penalty that grows logarithmically with the sample size N. As a result, BIC tends to favor simpler models than AIC, especially when N is large ([4]).

In our experiments, both criteria were applied to Lasso and Ridge regression for model selection, as shown in Figure 4a4b. The regularization parameters selected by BIC ($\alpha^{Lasso} = 0.0191$, $\alpha^{Ridge} =$

546.2277) are larger than that selected by AIC ($\alpha^{Lasso} = 0.0100$, $\alpha^{Ridge} = 17.8865$), which is consistent with the theoretical expectation that BIC prefers sparser, lower-complexity models.



(a) Information-criterion for Lasso model selection. Best α selected by AIC and BIC are 0.0100 and 0.0191, respectively.



(b) Information-criterion for Ridge model selection. Best α selected by AIC and BIC are 17.8865 and 546.2277, respectively.

3.3 Cross Validation for Robust Model Selection

To fully utilize the training data and generate more robust predictions, cross validation was adopted. Naive cross validation implementation doesn't explicitly preserve the percentage of each class, causing rare classes shown in figure 5 under-sampled and possibly not sampled at all. Thus, **Stratified K-Fold Cross Validation** was adopted for model assessment.

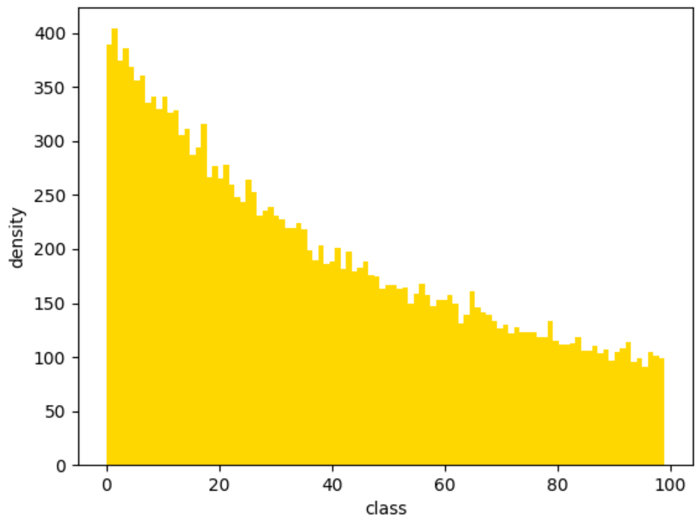


Figure 5: Histogram of Train Data Class Distribution. The most frequent class is class 1 (404/19573), the least frequent class is class 96 (91/19573). The class frequencies decrease almost monotonically.

3.4 Macro F1-Score for Balanced Model Prediction

To guarantee that models predict accurately across all classes, Macro F1-Score was used. This metric penalizes the model if it performs poorly on any single class, regardless of how small that class is. More precisely, Macro F1-Score measures the average of F1 scores that are calculated separately for each class, therefore preventing models from optimizing only for the majority classes. For both LR and SVM, we grid-searched the hyperparameter space using the above metrics. Results showed that for LR, using Ridge Regression with $\lambda = 0.011$ is the best choice, as for SVM, setting $C = 1$ and using Radial Basis Function (**RBF**) kernel yielded the highest score.

4 Model Development

Using only the basic matrix-computation library numpy, I had implemented multi-class Logistic Regression and Support Vector Machine. Both self-developed models demonstrated competitive accuracy on the test dataset compared with the scikit-learn library models. Further regulation techniques were implemented as well, including L1-norm and L2-norm penalty.

4.1 Multiclass Logistic Regression Development

In class, we had learned binary classification using logistic regression, here the task is generalized to K classes, where $K = 100$, and the input lies in $\mathbf{R}^d$. I will first show the derivation of the object function and the update rules, and then share some implementation details.
We first consider the unregularized case. Given an input vector $\mathbf{x} \in \mathbb{R}^d$, Multi-Class Logistic Regression models the conditional probability of the class label $y \in \{1, \dots, K\}$ using the softmax function:

$$P(y = k \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x} + b_k)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^T \mathbf{x} + b_j)} \tag{3}$$

where $\mathbf{w}_k \in \mathbb{R}^d$ and $b_k \in \mathbb{R}$ are the weights and bias for class k.

Our goal is to maximize the likelihood of the training data $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$. To ensure numerical stability and simplify optimization, we equivalently minimize the negative log-likelihood (Cross-Entropy Loss):

$$\min_{W,\mathbf{b}} J(W, \mathbf{b}) = -\frac{1}{N} \sum_{i=1}^{N} \log P(y = y^{(i)} \mid \mathbf{x}^{(i)}) \tag{4}$$

Using the indicator function $\mathbb{I}(\cdot)$, this can be rewritten as:

$$J(W, \mathbf{b}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbb{I}(y^{(i)} = k) \log P(y = k \mid \mathbf{x}^{(i)}) \tag{5}$$

Let $W \in \mathbb{R}^{K \times d}$ be the weight matrix where the k-th row is $\mathbf{w}_k^T$. We define the ground truth matrix $Y \in \mathbb{R}^{K \times N}$ and the probability matrix $P \in \mathbb{R}^{K \times N}$ such that:

$$Y_{ki} = \mathbb{I}(y^{(i)} = k) \tag{6}$$

$$P_{ki} = P(y = k \mid \mathbf{x}^{(i)}) \tag{7}$$

Let $X \in \mathbb{R}^{N \times d}$ be the design matrix containing input vectors as rows. The gradient of the loss function $J(W)$ with respect to W is given by:

$$\nabla_W J(W) = \frac{1}{N} (P - Y) X \tag{8}$$

We extend the objective function to include regularization terms to prevent overfitting.

L1 Regularization (Lasso): The loss function becomes:

$$J_1(W) = J(W) + \lambda_1 \|W\|_1 \tag{9}$$

The corresponding gradient update is:

$$\nabla_W J_1(W) = \nabla_W J(W) + \lambda_1 \operatorname{sgn}(W) \tag{10}$$

L2 Regularization (Ridge): The loss function becomes:

$$J_2(W) = J(W) + \frac{\lambda_2}{2}\|W\|_F^2 \tag{11}$$

where $\|W\|_F$ is the Frobenius norm. The corresponding gradient update is:

$$\nabla_W J_2(W) = \nabla_W J(W) + \lambda_2 W \tag{12}$$

Note that we **DO NOT penalize the intercepts** ([5]), since penalizing on the one-versus-rest decision boundaries will only reduce model accuracy.
One important observation concerns the appropriate scaling of the regularization hyperparameters for L1 and L2 penalties. When using L2 regularization, we selected $\lambda_2 = 0.01$, which yielded a local test accuracy of $\mathbf{89.18}\%$. However, naively setting the L1 regularization strength to the same value, $\lambda_1 = \lambda_2 = 0.01$, resulted in a substantially lower accuracy of only $\mathbf{78.17}\%$.
This discrepancy does not indicate an inherent inferiority of L1 regularization, but rather reflects the fundamentally different ways in which L1 and L2 penalties interact with the optimization landscape. The L2 penalty $\|W\|_F^2$ is smooth and distributes shrinkage continuously across all parameters, whereas the L1 penalty $\|W\|_1$ introduces a non-smooth geometry that encourages exact sparsity by driving many coefficients to zero. As a result, L1 regularization typically requires a **much smaller** regularization coefficient than L2 regularization to avoid excessive shrinkage and underfitting.
Empirically, the unregularized and L2-regularized solutions exhibit similar parameter magnitudes, with weights lying approximately in the range $[-0.035, 0.202]$ and $[-0.032, 0.189]$, respectively. Applying an L1 penalty of the same numerical scale therefore imposes a disproportionately strong constraint on the optimization problem, leading to premature coefficient elimination and underfitting.
These observations suggest that regularization strengths for L1 and L2 penalties are not directly comparable by numerical value. A practical rule of thumb is that L1 regularization coefficients should be tuned on a significantly smaller scale than their L2 counterparts, and that the appropriate magnitude of each penalty should be determined independently via cross validation rather than by direct substitution.

4.2 Support Vector Machine Development

Support vector machine uses a linear score function for each class k:

$$f_k(x) = w_k^T x + b_k \tag{13}$$

The objective is to minimize the **Regularized Multi-Class Hinge Loss** (specifically the formulation by Weston and Watkins ([12])). The total cost function $J(W, b)$ is defined as:

$$J(W,b) = \underbrace{\frac{1}{2}\sum_{k=1}^{K}\|w_k\|^2}_{\text{L2 Regularization}} + \underbrace{\frac{C}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f_j(x_i) - f_{y_i}(x_i) + \Delta)}_{\text{Average Hinge Loss}} \tag{14}$$

where:

- $W \in \mathbb{R}^{D\times K}$ is the weight matrix.
- $b \in \mathbb{R}^K$ is the bias vector.
- C is the regularization strength hyperparameter.
- Δ is the margin hyperparameter (default is 1.0).

Gradient descent is used to minimize $J(W, b)$. First, we define an indicator for margin violations for each sample i and class $j \neq y_i$:

$$\mathbb{I}_{ij} = \begin{cases} 1 & \text{if } w_j^T x_i + b_j - (w_{y_i}^T x_i + b_{y_i}) + \Delta > 0 \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

Let $P_i = \sum_{j\neq y_i} \mathbb{I}_{ij}$ be the count of classes that violate the margin for sample i. We construct a coefficient matrix $A \in \mathbb{R}^{N\times K}$:

$$A_{ij} = \begin{cases} \mathbb{I}_{ij} & \text{if } j \neq y_i \\ -P_i & \text{if } j = y_i \end{cases} \tag{16}$$

The gradients of the cost function with respect to the parameters are:

$$\nabla_W J = W + \frac{C}{N} X^T A \tag{17}$$

$$\nabla_b J = \frac{C}{N} \sum_{i=1}^{N} A_i \quad \text{(sum over samples for each class)} \tag{18}$$

Similar to our Logistic Regression, intercept b is not penalized. our experiment results showed that my own SVM achieved competitive accuracy on all datasets compared to the sklearn library model.

5 Conclusion

5.1 Conclusion-1: Linear Classifiers are Still Relevant Today

Although complex non-linear architectures have been the focus in recent years, linear classifiers are still some of the most robust and powerful models. By taking advantage of the sparsity of data, I discovered that the highly non-linear decision boundaries provided by neural networks may be powerful for complex tasks like vision and language processing, they are too noisy for this simple classification task. Simple models such as logistic regression and support vector machine are powerful enough. The rule of thumb here is to use the **simplest model assumption**, and **look carefully at the data** before delving straight into developing fancy models.

5.2 Conclusion-2: Ensembling Multiple Models Further Improves Prediction Performance

As my experiments had demonstrated, there are two key ways to ensemble models: 1) Train various **different** models and vote to decide the final decision. 2) Train the **same** models multiple times (possibly on different splits of train data), and average their final decisions. Both ways have their own advantages.
For method 1), **ensembling** tries to **combine the advantages** of all the adopted models, and generalizes better in different settings. The spirit of this method can still be seen in **Mixture of Experts** (**MoE**) ([10]), which trains multiple independent sub-networks that learn tasks in different professions.
As for method 2), we know from central limit theorem that **bagging** will reduce **prediction variance**. Since single decision tree is sensitive to the distribution of features,as our experiment results had shown, using random forests will significantly improve prediction stability.

References

[1] Hirotugu Akaike. Factor analysis and aic. *Psychometrika*, 52(3):317–332, 1987.

[2] Taher Al-Shehari and Rakan A. Alsowail. An insider data leakage detection using one-hot encoding, synthetic minority oversampling and machine learning techniques. *Entropy*, 23(10), 2021.

[3] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] Kenneth P Burnham and David R Anderson. Multimodel inference: understanding aic and bic in model selection. *Sociological methods & research*, 33(2):261–304, 2004.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[6] Ian Jolliffe. *Principal Component Analysis*. John Wiley & Sons, Ltd, 2014.

[7] Henry F Kaiser. The application of electronic computers to factor analysis. *Educational and Psychological Measurement*, 20(1):141–151, 1960.

[8] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[9] Gideon Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.

[10] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V Le, Geoffrey E Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[11] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

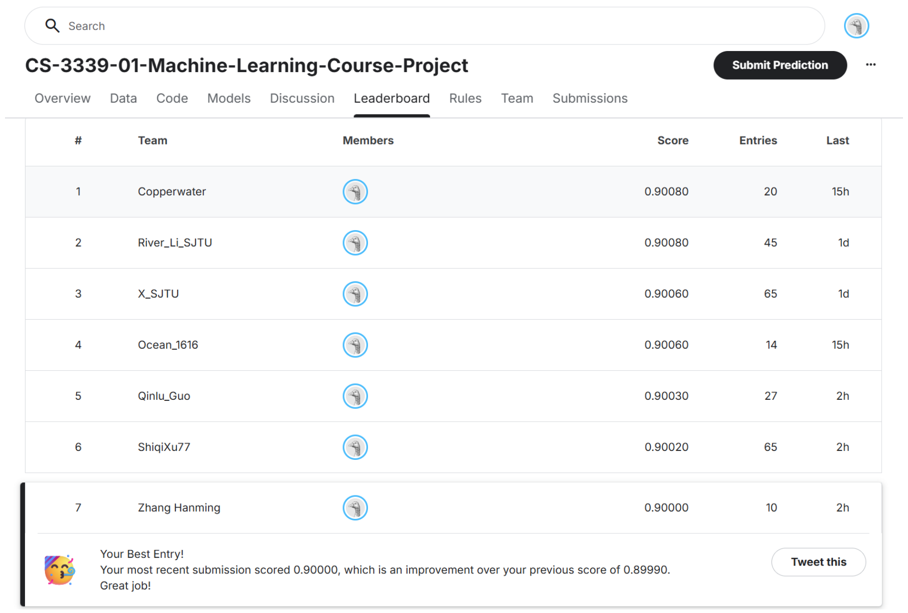
[12] Jason Weston and Chris Watkins. Support vector machines for multi-class pattern recognition. *Proceedings of ESANN*, 1999.

A Kaggle Test Results

Model	Accuracy(%)
Random Forest (1850 estimators)	**90.00**
Random Forest (1700 estimators)	89.95
Random Forest (1600 estimators)	89.92
Random Forest (1500 estimators)	89.89
Random Forest (1400 estimators)	89.84
Random Forest (1300 estimators)	89.87
Random Forest (1200 estimators)	89.88
Random Forest (1200 estimators)	89.88
Random Forest (1100 estimators)	89.87
Random Forest (1000 estimators)	89.90
Random Forest (900 estimators)	89.82
Random Forest (800 estimators)	89.84
Random Forest (700 estimators)	89.79
Random Forest (600 estimators)	89.88
Random Forest (500 estimators)	89.86
Decision Tree (gini)	84.41
Decision Tree (entropy)	84.24
Decision Tree (log_loss)	84.22
SVM (C=0.01)	89.67
SVM (C=0.01 50d)	89.70
SVM (C=0.01 40d)	89.73
SVM (C=0.01 30d)	89.57
SVM (C=0.01 20d)	89.55
MLP (hidden_layer_shape=[72, 24])	89.41
MLP (hidden_layer_shape=[72, 24] 40d)	87.96
Lasso (alpha=0.01)	89.08
Lasso (alpha=0.01 30d)	89.57
Lasso (alpha=0.01 20d)	89.55
Lasso (alpha=0.01 10d)	89.02
Ridge (alpha=0.1)	89.15
Ridge (alpha=0.1 30d)	89.70
Ridge (alpha=0.1 20d)	89.65
Ridge (alpha=0.1 10d)	89.18

Table 1: Model Accuracy Comparison

B Best Test Result Screenshot



CS-3339-01-Machine-Learning-Course-Project

#	Team	Members	Score	Entries	Last
1	Copperwater		0.90080	20	15h
2	River_Li_SJTU		0.90080	45	1d
3	X_SJTU		0.90060	65	1d
4	Ocean_1616		0.90060	14	15h
5	Qinlu_Guo		0.90030	27	2h
6	ShiqiXu77		0.90020	65	2h
7	Zhang Hanming		0.90000	10	2h

Your Best Entry!
Your most recent submission scored 0.90000, which is an improvement over your previous score of 0.89990.
Great job!

Figure 6: Best Prediction Result on Kaggle (my user name is **Zhang Hanming**)