Statistical Learning and Inference Project Report

Fuming Zhang, 118033910025

1 Introduction

In recent years, with the rapid development of deep learning, researchers have made significant progress in the fields of image recognition and natural language processing. Although deep neural networks can achieve better results in specific areas, their interpretability is still unsatisfactory. And the deeper the network, the harder it is to train and the more time it takes. Compared with the neural networks, the classical methods in statistical learning and inference are efficient and straightforward.

In this project, we are going to use some of the classical methods in statistical learning to solve the image classification problem under ImageNet. Although the principles of these methods are concise, it is quite complicated to implement from scratch. With the help of scikit-learn [1], we can focus on the adjustment of the hyperparameters on the method and the comparison between different methods. We not only tried various classification methods such as ridge classification, logistic regression, linear discriminant analysis, support vector machine and k nearest neighbour classification, but also tried to perform data preprocessing and dimensionality reduction. After we augment the training set using a semi-supervised learning method, we selected the best model by k-fold cross-validation, we achieved an accuracy of 0.93754 on the test set and ranked 37th on the private leaderboard.

The remainder of this paper is organized as follows. In Section 2, we describe the dataset this project and some related processing methods. Two dimensionality reduction methods are introduced in Section 3. Then, the five classification methods we used is illustrated in Section 4. We discuss the model selection approach in Section 5. Evaluation of all the above methods is described in Section 6. Finally, we conclude the paper in Section 7.

2 Data

In this section, we will first provide a overview of the dataset in 2.1, then we will introduce data preprocessing and augmentation in 2.2 and 2.3, respectively.

2.1 Dataset Overview

In this project, data is collected from ImageNet, which contains 12 categories for classification. The dataset is split into training set and testing set. The training set contains 7800 images (650 images for each class), while the test set contains 15600 images (1300 images per class). The 4096 features provided are generated by a neural network trained on ImageNet.

By observing the training set, it is not difficult to find that each data has a value of 0 in many dimensions, and the values in all dimensions are greater than or equal to zero.

2.2 Preprocessing

Standardization of datasets is a common requirement for many machine learning estimators. Otherwise, they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance. In practice we often ignore the shape of the distribution and just transform the data to center it by removing the mean value of each feature, then scale it by dividing non-constant features by their standard deviation.

For instance, many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the l1 and l2 regularizers of linear models) assume that all features are centered around zero and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

Therefore, we mainly use min-max scaling, l1 and l2 normalization these three preprocessing methods in this project. As for min-max scaling, we use the implementation provided by 'sklearn.preprocessing.MinMaxScaler' to scale the data to the $[0, 1]$ interval. For l1 and l2 normalization, we use the implementation provided by 'sklearn.preprocessing.normalize'. By specifying the value of parameter 'norm' as 'l1' or 'l2', we scale input vectors individually to unit norm. The results of these preprocessing methods are compared in 6.3.

2.3 Augmentation

Limited by the number of training samples, after a series of hyperparameter and preprocessing adjustment model selection, our prediction accuracy is difficult to further improve, as if hit the accuracy wall. To solve this problem, we decide to add more data to the training set. There are many ways to

- *Private leaderboard accuracy 0.93754 with rank 37*
- *Public leaderboard accuracy 0.94401 with rank 19*
- *E-mail: zhangfuming-alex@sjtu.edu.cn*

augment existing datasets and produce more robust models. In the image domain, these are done to utilize the full power of the convolutional neural network, which is able to capture translational invariance. However, in this project, also limited by the number of samples, we have not tried to use the neural network method.

Considering that the given data is already in the form of generated features, we cannot perform transformations and color distortions to the data like images, but we can add some random noise to the data. However, we did not use the above method in practice. Instead, we choose the semi-supervised learning method which adds the same results predicted on the test set by several classifiers into the training set. By repeating this approach, we achieved the highest prediction accuracy of 93.754% with ridge classifier on the private leaderboard.

3 Dimensionality Reduction

In this section, we will introduce two dimensionality reduction methods, principle component analysis in 3.1 and factor analysis in 3.2.

3.1 Principle Component Analysis

Principle Component Analysis (abbreviation as PCA) is used to decompose a multivariate dataset in a set of successive orthogonal components that explain a maximum amount of the variance. Given a set of data in IR^p, denote the observations by $x_1, x_2, ..., x_N$, and consider the rank-q linear model for representing them

$$f(\lambda) = \mu + V_q \lambda, \tag{1}$$

where μ is a location vector in IR^p, V_q is a $p \times q$ matrix with q orthogonal unit vectors as columns, and λ is a q vector of parameters. Fitting such a model to the data by least squares amounts to minimizing the reconstruction error

$$\min_{\mu, \{\lambda_i\}, V_q} \sum_{i=1}^{N} ||x_i - \mu - V_q \lambda_i^2|. \tag{2}$$

We can partially optimize for μ and the λ_i to obtain

$$\hat{\mu} = \bar{x}, \tag{3}$$
$$\hat{\lambda}_i = V_q^T (x_i - \bar{x}). \tag{4}$$

This leaves us to find the orthogonal matrix V_q:

$$\min_{V_q} \sum_{i=1}^{N} ||(x_i - \bar{x}) - V_q V_q^T (x_i - \bar{x})||^2. \tag{5}$$

The $p \times p$ matrix $H_q = V_q V_q^T$ is a projection matrix, and maps each point x_i onto its rank-q reconstruction $H_q x_i$, the orthogonal projection of x_i onto the subspace spanned by the columns of V_q.

3.2 Factor Analysis

The classical factor analysis model has the form

$$X = AS + \epsilon. \tag{6}$$

Here S is a vector of $q < p$ underlying latent variables or factors, A is a $p \times q$ matrix of factor loadings, and the ϵ_j are uncorrelated zero-mean disturbances. Typically the S_l and the ϵ_j are modeled as Gaussian random variables, and the model is fit by maximum likelihood. The parameters all reside in the covariance matrix

$$\Sigma = AA^T + D_\epsilon, \tag{7}$$

where $D_\epsilon = \mathrm{diag}[\mathrm{Var}(\epsilon_1), ..., Var(\epsilon_p)]$. The columns of A are referred to as the factor loadings, and are used to name and interpret the factors.

Factor analysis can produce similar components (the columns of its loading matrix) to PCA. However, one can not make any general statements about these components. The main advantage for Factor Analysis over PCA is that it can model the variance in every direction of the input space independently (heteroscedastic noise). We followed the instructions given in [2] to find the best n_components of our data set in this project and the results are shown in 6.2.

4 Classification Methods

In this section, we will present the classification methods we used in this project. For each method, we first review its definition and basic principles, and then introduce how we use scikit-learn [1] based implementations.

4.1 Ridge Regression

Compared with the ordinary least squares method, ridge regression shrinks the regression coefficients by imposing a penalty on their size. The ridge coefficients minimize a penalized residual sum of squares,

$$\hat{\beta}^{\mathrm{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}. \tag{8}$$

Here $\lambda \geq 0$ is a complexity parameter that controls the amount of shrinkage; the larger the value of λ, the greater the amount of shrinkage. The coefficients are shrunk toward zero (and each other) [3].

In this project, we use the implementation provided by 'sklearn.linear_model.RidgeClassifier'. It is a classifier using ridge regression. When dealing with a multi-class classification task, n_class classifiers are trained in a one-versus-all approach. Concretely, this is implemented by taking advantage of the multi-variate response support in ridge. Some important parameters of this method are introduced as follows:

- **alpha**: It is a positive float indicating the regularization strength. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Large values specify stronger regularization. It is worth noting that the alpha here corresponds to

C^{-1} in other linear models such as logistic regression and linear SVC.

- **normalize**: This is a boolean value which indicates whether to perform l2 normalization on the data. Its default value is false, and we have not used this parameter in our experiments. When we decide to perform data preprocessing, we will process the data before the model is trained.
- **class_weight**: The value of this parameter should be a dictionary in Python or 'balanced', which denotes the weights associated with classes. If not given, all classes are supposed to have weight one. If 'balanced' mode is used, the weights are adjusted inversely proportional to class frequencies as $\frac{\text{n\_samples}}{\text{n\_classes} \times \text{np.bincount(y)}}$. Since the amount of data in each class is the same in our training set, the effect of using the 'balanced' mode and the default unspecified parameter is the same.
- **solver**: The candidate values for this parameter are 'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag' and 'saga'. It represents the method used in the computational routine. The details of these methods will not be discussed here, but we will compare the results of different methods in 6.3.

4.2 Logistic Regression

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression models are usually fit by maximum likelihood, using the conditional likelihood of G given X. Since $Pr(G|X)$ completely specifies the conditional distribution, the multinomial distribution is appropriate. The log-likelihood for N observations is

$$l(\theta) = \sum_{i=1}^{N} \log p_{g_i}(x_i; \theta), \tag{9}$$

where $p_k(x_i; \theta) = Pr(G = k|X = x_i; \theta)$. As an optimization problem, binary class L2 penalized logistic regression minimizes the following cost function:

$$\min_{\omega,c} \frac{1}{2}\beta^T\beta + C\sum_{i=1}^{N} \log(\exp(-y_i(\beta_i^T X_i + c)) + 1). \tag{10}$$

In the above notation, it's assumed that the observation y_i takes values in the set $-1, 1$.

In this project, we use the implementation provided by 'sklearn.linear_model.LogisticRegression'. Table 1 summarizes the penalties supported by different solvers in logistic regression. Some important parameters of this method are introduced as follows:

- **penalty**: This parameter is used to specify the norm used in the penalization and the candidate values are 'l1' or 'l2'.
- **dual**: It is a boolean value which indicates whether to solve the principle or dual formulation problem. Dual formulation is only implemented for l2 penalty with liblinear solver. The implementer of this method recommends that we'd better set dual=False when n_samples $>$ n_features.
- **C**: The value of this parameter must be a positive float, which represents the inverse of regularization strength. Different from the implementation of ridge regression as we have mentioned in Subsection 4.1, smaller values specify stronger regularization.
- **class_weight**: This parameter is used in the same way as described in Subsection 4.1.
- **solver**: The candidate values for this parameter are 'newton-cg', 'lbfgs', 'liblinear', 'sag' and 'saga'. It represents the algorithm used in the optimization problem. Regarding these algorithms, the following points need to be emphasized:
    1) For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones.
    2) For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss; 'liblinear' is limited to one-versus-rest schemes.
    3) 'newton-cg', 'lbfgs' and 'sag' only handle L2 penalty, whereas 'liblinear' and 'saga' handle L1 penalty.
- **multi_class**: This parameter denotes the Classification method. Candidate values are 'ovr', 'multinomial' and 'auto'. If the option chosen is 'ovr', then a binary problem is fit for each label. For 'multinomial', the loss minimized is the multinomial loss fit across the entire probability distribution.

4.3 Linear Discriminant Analysis

Linear discriminant analysis (abbreviation as LDA) is a classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule. The model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix. The linear discriminant function has the form,

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k. \tag{11}$$

The parameters of the Gaussian distributions can be estimated as follows:

- $\hat{\pi}_k = \frac{N_k}{N}$, where N_k is the number of class-k observations;
- $\hat{\delta}_k = \sum_{g_i=k} \frac{x_i}{N_k}$;
- $\hat{\Sigma} = \sum_{k=1}^{K} \sum_{g_i=k} \frac{(x_i-\hat{\mu}_k)(x_i-\hat{\mu}_k)^T}{N-K}$.

LDA is attractive because it has closed-form solution that can be easily computed, is inherently multiclass, has proven to work well in practice, and has no hyperparameters to tune.

In this project, we use the implementation provided by 'sklearn.discriminant_analysis.LinearDiscriminantAnalysis'. Some important parameters of this method are introduced as follows:

- **priors**: It is an array of the form (n_classes,) which represents the prior probability of each class. There are 12 categories in our data set, clearly indicating that the prior probability of each category is $\frac{1}{12}$ does not help the classification results.

TABLE 1
Penalties supported by different solvers in Logistic Regression

Penalties	liblinear	lbfgs	newton-cg	sag	saga
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes

- **solver**: The candidate values for this parameter are 'svd', 'lsqr' and 'eigen', which indicates the method used to solve the problem. The default solver is 'svd'. It can perform both classification and transform, and it does not rely on the calculation of the covariance matrix. This can be an advantage in situations where the number of features is large. However, the 'svd' solver cannot be used with shrinkage. The 'lsqr' solver is an efficient algorithm that only works for classification. It supports shrinkage. The 'eigen' solver is based on the optimization of the between-class scatter to within class scatter ratio. It can be used for both classification and transform, and it supports shrinkage. However, the 'eigen' solver needs to compute the covariance matrix, so it might not be suitable for situations with a high number of features. We will discuss the performance of these methods in 6.5.

4.4 Support Vector Machine

Support Vector Machine (abbreviation as SVM) constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification and regression. Given slack variables ξ, the formulation of "standard" support vector classifier is:

$$\min_{\beta,\beta_0} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}\xi_i \qquad (12)$$
$$\text{subject to } \xi_i \geq 0, y_i(x_i^T\beta + \beta_0) \geq M(1-\xi_i), \forall i.$$

The Lagtange (primal) function is

$$L_P = \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\mu_i\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(x_i^T\beta+\beta_0) - (1-\xi_i)], \qquad (13)$$

which we minimize w.r.t β, β_0 and ξ_i. Setting the respective derivatives to zero, we get

$$\beta = \sum_{i=1}^{N}\alpha_i y_i x_i, \qquad (14)$$

$$0 = \sum_{i=1}^{N}\alpha_i y_i, \qquad (15)$$

$$\alpha_i = C - \mu_i, \forall i, \qquad (16)$$

as well as the positivity constraints $\alpha_i, \mu_i, \xi_i \geq 0\ \forall i$. By substituting Eq.(14)-(16) into Eq.(13), we obtain the Lagrangian dual objective function

$$L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_{i'}y_i y_{i'}x_i^T x_{i'}, \qquad (17)$$

which gives a lower bound on the objective function for any feasible point. We maximize L_D subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{N}\alpha_i y_i = 0$.

In this project, we use the implementation provided by 'sklearn.svm.LinearSVC'. Similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. Some important parameters of this method are introduced as follows:

- **penalty**: This parameter is used to specify the norm used in the penalization and the candidate values are 'l1' or 'l2'.
- **loss**: This parameter specifies the loss function. 'hinge' is the standard SVM loss while 'squared_hinge' is the square of the hinge loss.
- **dual**: It specifies whether to solve the dual or primal optimization problem. Prefer dual=False when n_samples > n_features.
- **C**: This is the penalty parameter of the error term.

4.5 K Nearest Neighbour

neighbours-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbours of each point: a query point is assigned the data class which has the most representatives within the nearest neighbours of the point.

The k-neighbours classification is the most commonly used technique. The optimal choice of the value k is highly data-dependent: in general a larger k suppresses the effects of noise, but makes the classification boundaries less distinct. Specifically, the k-nearest neighbour fit has the follow formulation:

$$\hat{f}(x_0) = \frac{1}{k}\sum_{l=1}^{k} f(x_{(l)}), \qquad (18)$$

where the subscripts in parantheses (l) indicate the sequence of nearest neighbours to x_0.

In this project, we use the implementation provided by 'sklearn.neighbors.KNeighborsClassifier'. Some important parameters of this method are introduced as follows:

- **n_neighbours**: This parameter specifies the number of neighbours to use by default for k-neighbours query.
- **weights**: It represents the weight function used in prediction. Two candidate values are 'uniform' and 'distance'. For 'uniform', all points in each neighborhood are weighted equally. For 'distance', points are weighted by inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- **p**: It indicates the power parameter for the Minkowski metric. Usually p is equal to 1 or 2.
- **algorithm**: This parameter denotes the algorithm used to compute the nearest neighbours. Candidate algorithms are 'ball_tree', 'kd_tree' and 'brute'.
- **leaf_size**: This parameter is passed to BallTree or KDTree, which can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

5 Model Selection

When evaluating different hyperparameters for estimators, such as the 'C' parameter which must be manually set for an SVM, we perform k-fold cross-validation (abbreviation as CV). In the basic approach, the training set is split into k smaller sets. The following procedure is followed for each of the k "folds":

1) A model is trained using of the folds as training data.
2) The resulting model is validated on the remaining part of the data.

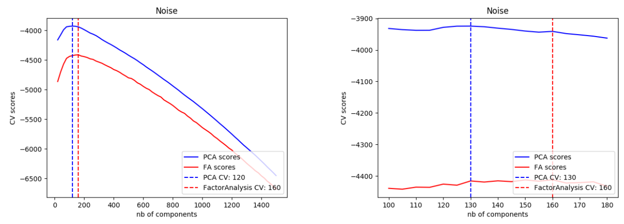
The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but is suitable for the classification task with a small number of samples in this project.

We mainly use the function 'sklearn.model_selection.cross_val_score' to evaluate the performance of the estimator in different hyperparameter configurations. As for the partition strategy, we use 'sklearn.model_selection.StratifiedKFold', where the folds are made by preserving the percentage of samples for each class. We set the value of parameter 'n_splits', which stands for the number of folds, to 5. In order to make the partition result reproducible, we fixed the value of parameter 'random_state' to 9.

In order to conveniently compare the performance of estimators in different hyperparameter configurations, we also use the grid search mechanism provided by 'sklearn.model_selection.GridSearchCV', which exhaustively generates candidates from a grid of parameter values specified with the 'param_grid' parameter.

6 Evaluation

In this section, we present our evaluation of all of the above methods, and the accuracy in the evaluation is based on the 5-fold cross-validation on the training set unless otherwise stated.



(a) Coarse-Grained (b) Fine-Grained

Fig. 1. PCA

6.1 Dataset Augmentation

As we mentioned in 2.3, our best model is obtained by training on the augmented dataset. In practice, we combined the prediction results of ridge classification, logistic regression and linear SVM these three classifiers. As this process is repeated, the results are shown in Table 2.

6.2 PCA

As we have mentioned in 3.2, following the instructions given in [2], we draw the results as shown in Figure 1. In Figure 1(a), we set the dimension range from 20 to 1500 with a span of 20. In Figure 1(b), we set the dimension range from 100 to 180 with a span of 5.

6.3 Ridge Classification

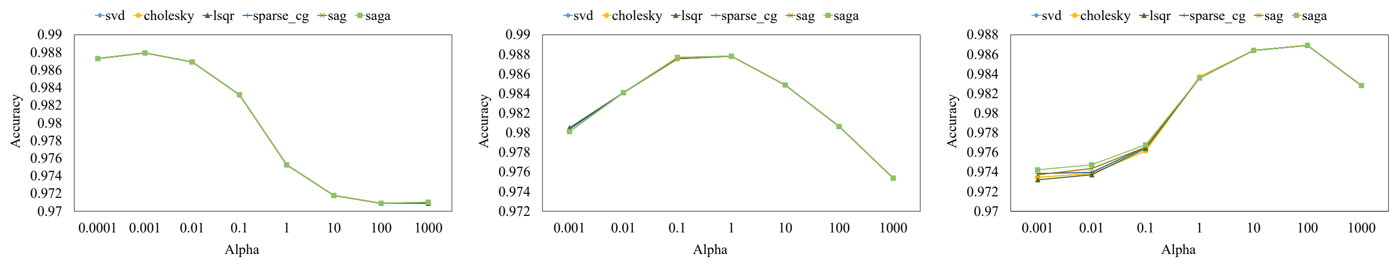
In this subsection, we compare the prediction accuracy under different solvers by setting different alpha values under different preprocessing. The experiment results are drawn in Figure 2. Basically, different solvers have little effect on prediction accuracy. In Figure 2(a), we performed l1 normalization on the data and the prediction accuracy reaches a maximum of 0.98795 at alpha=0.001. In Figure 2(a), we performed l2 normalization on the data and the prediction accuracy reaches a maximum of 0.98782 at alpha=1. In Figure 2(c), we performed min-max scaling on the data and the prediction accuracy reaches a maximum of 0.98692 at alpha=100. The best performance of this model submitted to Kaggle is 0.92994 (in private leaderboard).

6.4 Logistic Regression

Fix the value of the parameter 'multi_class' as 'ovr', we compare the performance under different solvers by setting different C under different penalty. The experiment results are drawn in Figure 3. Set the penalty as 'l1', with C equal to 0.1, both valid solvers saga and liblinear have achieved the highest prediction accuracy of 0.98385 and 0.9841, respectively. Set the penalty as 'l2', solver newton-cg and lbfgs

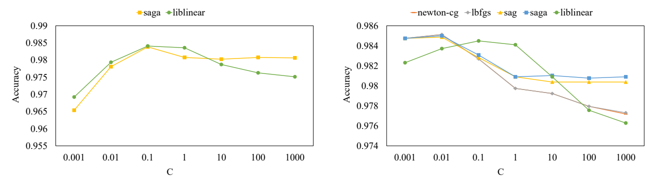
TABLE 2
The Prediction Results with Dataset Augmentation

Dataset	Private Leaderboard Accuracy
Original	0.92994
Augmentation Iter 1	0.93708
Augmentation Iter 2	0.93727
Augmentation Iter 3	0.93754



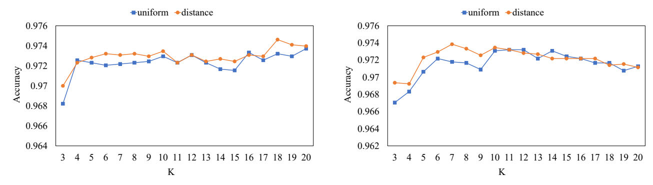
(a) Pre=L1 (b) Pre=L2 (c) Pre=Min-Max

Fig. 2. Ridge Classification



(a) Penalty=L1 (b) Penalty=L2

Fig. 3. Logistic Regression



(a) Power=1 (b) Power=2

Fig. 6. KNN

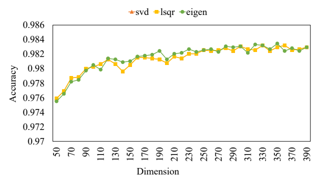


Fig. 4. LDA

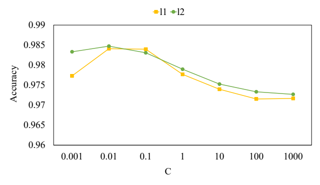


Fig. 5. SVM

reach the highest prediction accuracy of 0.98513 at C=0.01. The best performance of this model submitted to Kaggle is 0.92353 (in private leaderboard).

6.5 LDA

When using LDA without performing dimensionality reduction of the original data first, the program will give a warning that variables are collinear and the prediction accuracy is low. So, in this subsection, using PCA to reduce the original data to different dimensions, we show the performance of LDA with different solvers in Figure 4. When the dimension of the data is not reduced, the prediction accuracy of LDA is only 0.97346, which is lower than the accuracy when the data is reduced to 50 dimensions in the figure.

6.6 SVM

Figure 5 shows the prediction accuracy of linear svc under different C when l1 and l2 are used as penalty separately. With penalty as l1, it achieves highest prediction accuracy 0.9841 when C equals to 0.01. Similarly, with penalty as l2, the highest prediction accuracy is 0.98474 when C equals to 0.01. The best performance of this model submitted to Kaggle is 0.92847 (in private leaderboard).

6.7 KNN

Setting the power parameter of Minkowski metric to 1 and 2, respectively, we plot the prediction accuracy under different K values with weights set to uniform and distance in Figure 6. For power equals to 2, when we choose distance mode as weight function, the estimator reaches the highest prediction accuracy of 0.97385 at K=7. For power equals to 1, the estimator reaches the highest prediction accuracy of 0.97462 at K=18.

7 Conclusion

In this paper, we have used ridge classification, logistic regression, linear discriminant analysis, support vector machine and k nearest neighbour classification these five approaches to perform image classification. We have also tried data preprocessing and dimensionality reduction. When we hit the accuracy wall, we propose to use the semi-supervised learning method to augment the dataset to further improve the prediction accuracy of the model. And finally we achieved an accuracy of 0.93754 on the test set and ranked 37th on the private leaderboard.

Acknowledgments

The author would like to thank Prof. Liqing Zhang and TA Jianfu Zhang and Xiang Li for their instructions on this work.

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[2] A. Gramfort and D. Engemann, "Model selection with probabilistic pca and factor analysis," https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_fa_model_selection.html#sphx-glr-auto-examples-decomposition-plot-pca-vs-fa-model-selection-py, accessed January 18, 2019.

[3] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, NY, USA:, 2001, vol. 1, no. 10.