Statistical Learning and Inference Project Report

Fuming Zhang, 118033910025

Abstract—Statistical Learning

Index Terms—Statistical learning

1 Introduction

With the help of scikit-learn [1], we ...

2 Data

2.1 Preprocessing

Scale, MinMaxScale, L1/2 Normalization

2.2 Augmentation

3 Dimensionality Reduction

PCA, Factor Analysis

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,

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

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

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 'balanceed', 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 discuessed here, but we will compare the results of different methods in Section **??**.

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

$$l(\theta) = \sum_{i=1}^{N} \log p_{g_i}(x_i; \theta), \tag{2}$$

- *E-mail: zhangfuming-alex@sjtu.edu.cn*

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:

$$\min_{\omega,c} \frac{1}{2}\beta^T\beta + C\sum_{i=1}^{N}\log(\exp(-y_i(\beta_i^T X_i + c)) + 1). \quad (3)$$

Similarly, L1 regularized logistic regression solves the following optimization problem:

$$\min_{\omega,c} ||\beta||_1 + C\sum_{i=1}^{N}\log(\exp(-y_i(\beta_i^T X_i + c)) + 1). \quad (4)$$

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 and 2 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,

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log\pi_k. \quad (5)$$

The parameters of the Gaussian distributions can be estimates 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.
- **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 Section **??**.

4.4 Support Vector Machine

4.5 K Nearest Neighbour

5 Model Selection

Cross Validation, Grid Search

6 Evaluation

7 Discussion

8 Conclusion

In this paper [2], we have proposed...

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

TABLE 2
Behaviors of different solvers in Logistic Regression

Behaviors	liblinear	lbfgs	newton-cg	sag	saga
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

Acknowledgments

The author would like to thank...

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] F. Zhang, Z. Tang, M. Chen, X. Zhou, and W. Jia, "A dynamic resource overbooking mechanism in fog computing," in *15th IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, 2018, pp. 89–97.