Midterm Project Report
CS 6350 Machine Learning

Ryan Dalby

26th October, 2021

Progress

Commit to reference for this report: `https://github.com/dalbyryan3/cs-6350-machine-learning/tree/6eca7e9345f2a286265fb92d21f3f69f0f92fdfa`

Overview

For my project I chose to do the competitive kaggle project. This project involves predicting if a person makes over 50k a year using data from the 1994 census. This machine learning problem can be considered supervised binary classification of tabular data. The attributes (features) of the data contain both continuous and discrete values.

For this project I primarily used scikit-learn [1], a Python library with implementations of common machine learning algorithms, many of which I have implemented or learned about in this class.

In order to create a model that could accurately predict income my initial steps involved data pre-processing to get the data in an easily trainable form. I explored different methods of encoding binary features and normalization of data values. I then trained and analyzed logistic regression with the data. Here I noted the differences between optimization methods and the effect of data normalization. Next, I explored using support vector machine based classification with a radial basis kernel. Lastly, I discuss my next steps which primarily involve investigating ensemble-based methods and more complicated models.

Data Pre-processing

My first step of this project was to get familiar with the data. The training data I was provided with contains approximately 25000 labelled examples and 14 features made up of 8 categorical features and 6 numerical features. My first task was to convert the 8 categorical features to numerical features that could be used with many common scikit-learn algorithms. There are many ways to encode discrete class values as numerical ones. The most straight forward way is called ordinal encoding or label encoding, this involves simply mapping a given class to an integer. This technique is simple but introduces undesirable artificial ordering into the data [2].

I chose to utilize one-hot encoding, a simple technique that overcomes the artificial ordering problems of label encoding [2]. This technique maps a given class of a feature to a unique binary vector representing the class. In practice this expands the number of features since every categorical feature becomes represented by new features representing all of the possible values it can take on. For this machine learning task, one-hot encoding expanded the number of features from 14 to 108.

Next, I randomly split the labelled data I was provided with into training and test sets using an 80% training, 20% test split. Some other things to note about this split are that I made sure to keep the test data completely isolated (i.e. I used the normalization scaling values of the training data only on the test data) and use the same encoding for training and test data. This split was an arbitrary choice but I will later discuss how this splitting affected the model performance and how using ensemble methods may help create a better model.

I will also later discuss how I investigated normalization data pre-processing which ended up having a major impact in both trainability and performance of logistic regression and support vector classification.

Logistic Regression Classification

After one-hot encoding and splitting the labelled data I was left with a training and test set that I could use to train various machine learning models. The first model I trained was logistic regression, a linear binary classification model whose objective is to maximize loss entropy. Logistic regression provides an interpretable probabilistic output that is generalized by a softmax function for more than 2 output classes [3]. The loss function I utilized has l_2 regression and is

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

where w is the weight vector, C is a regularization constant, c is a bias term, y_i is a vector of labels and X is a matrix of number of examples by number of features.

I first utilized a stochastic gradient descent solver but found that it wasn't consistently finding a good optima. I then decided to used a direct approach of finding a the optima using limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (lbfgs) which uses an estimated of the Hessian (convexity) to optimize rather than just the gradient (thus less hyperparameters).

The trained results are shown in Table 1 where the model achieved a test AUC of 0.631. The results can also be seen in Figure 1 which is the learning curve for logistic regression on the data.

Learning curves give a good indication of the bias and variance of a hypothesis by training on varying sizes of data and observing how the score/accuracy changes [4]. This learning curve shows that we have relatively small variance (training and validation scores are close to each other), but a large bias (since we want to get over .90 accuracy), implying we may need a more powerful model or more data.

I then implemented data normalization, which means transforming the values of each feature to be zero mean and unit variance. I then retrained a logistic regression model and got much better results as shown in Table 1, improving test AUC from .631 to .755.

There are some possible reasons that normalization of the data improved performance. First, having the data on the same scale implies that each feature is equally important, which is usually the assumption when we being to learn from data. Another possible reason for the improvement is that it allows easier optimization since the features are on the same magnitude scale. This can be seen by comparing the learning curve of the non-normalized data of Figure 1 to that of the normalized data in Figure 2. We see here that the learning curve with the normalized data is much more stable as we increase the number of examples.

Support Vector Machine Classification

Next I investigated using support vector machine based classification. Support vector classification (SVC) has an objective of maximizing the margin between a decision hyperplane and examples [5].

The objective can be stated as

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \quad \text{Subject to: } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \;\; \zeta_i \geq 0, i = 1, ..., n$$

where w is the weight vector, C is a regularization constant (inverse regularization parameter), ζ_i controls distance from boundary, b is a bias term, y_i is a vector of labels and x_i is an example. A kernel can be used to transform the data to be linear, a kernel is defined by $\phi(x_i)$ as $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$.

SVC with non-linear kernels can be used to learn non-linear relationships in data by transforming the data in a way that it can appear as linear to the SVC classifier. Initially, I trained an SVC with a linear kernel and found results equivalent or worse than logistic regression so I moved onto using a polynomial kernel where I got no better results. I then tried a radial basis kernel (rbf) and got comparable but slightly better results than logistic regression. The results for SVC with an rbf kernel can be seen in Table 1 where a tes AUC of 0.770 was achieved.

Looking at the learning curves for this model it is clear that this model overfits the training data to a higher degree than the logistic regression model. This can be seen in Figure 3 where the learning curve shows a high variance as evidenced by the gap between training and validation score. I experimented with different regularization values but found that the test AUC score did not change much, getting worse with less or more regularization.

Next Steps

After investigating logistic regression and support vector classification the next step would be to try an ensemble model of these classifiers. The reason this would be beneficial would be because I noticed that the train test splitting can affect the performance the model. Thus using a ensemble method, specifically bagging, would allow the best performance of logistic and support vector classification on the data.

I also would like to investigate another ensemble method called gradient boosting classification as it seems to show promise with tabular data classification in literature [6].

I also intend to research multi-layer perceptrons (fully-connected neural networks) to investigate their performance at this task. This has many more hyperparameters than the models investigated in this report so it will be important to explore the hyperparameter space.

In the end, I will attempt to get a test AUC near 0.90 by investigating new models and optimizing the hyperparameters of the most promising model, and creating a final ensemble model.

	Logistic	Logistic with normalized data	SVC with normalized data
Train accuracy	0.794	0.856	0.888
Train AUC	0.631	0.769	0.821
Test accuracy	0.790	0.843	0.849
Test AUC	0.631	0.755	0.770

Table 1: Performance of Tested Models

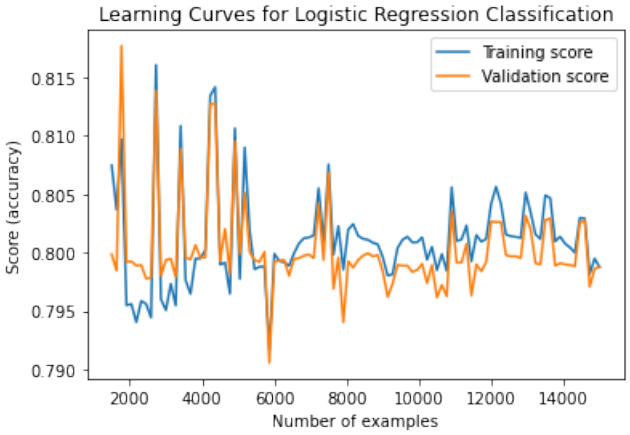

Figure 1: Learning Curve for Logistic Regression Model

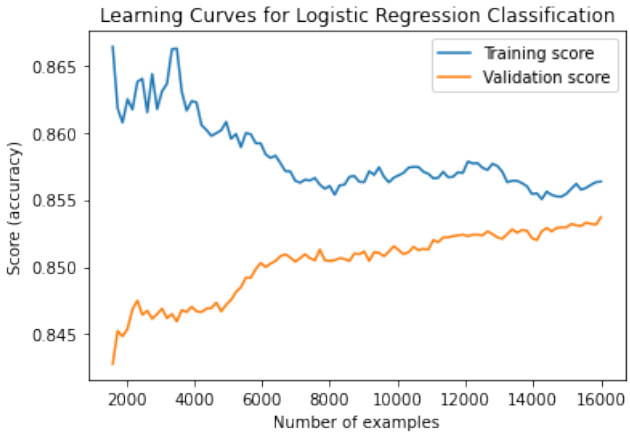

Figure 2: Learning Curve for Logistic Regression Model with Normalized Data

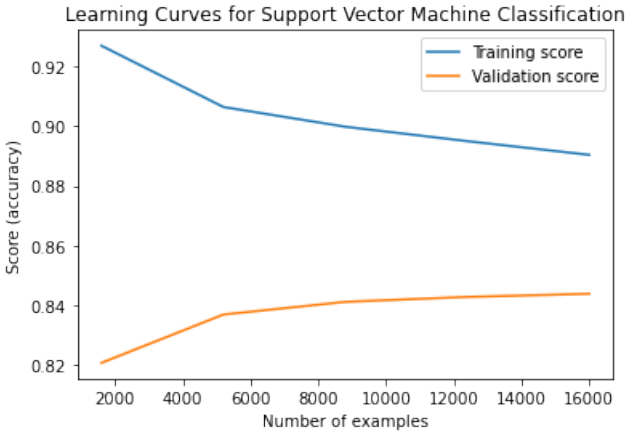

Figure 3: Learning Curve for Support Vector Classification with Normalized Data

References

[1] "scikit-learn: machine learning in Python — scikit-learn 1.0 documentation." [Online]. Available: https://scikit-learn.org/stable/

[2] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *Journal of Big Data*, vol. 7, no. 1, p. 28, Apr. 2020. [Online]. Available: https://doi.org/10.1186/s40537-020-00305-w

[3] "1.1. Linear Models." [Online]. Available: https://scikit-learn/stable/modules/linear_model.html

[4] A. Ng, "Advice for applying Machine Learning," p. 30.

[5] "1.4. Support Vector Machines." [Online]. Available: https://scikit-learn/stable/modules/svm.html

[6] "1.11. Ensemble methods." [Online]. Available: https://scikit-learn/stable/modules/ensemble.html