Predicting Income Potential

Elan Miller, Louis Schlessinger

1 Introduction

Predicting income potential is an old problem that has many applications. For example, income prediction is relevant for pension plan valuation. Additionally, Kibekbaev and Duman briefly summarize the importance of income forecasting for credit card companies:

Individual consumer income information enables the banks to validate application data, target new prospects, and segment existing customers. Income can be understood as the summation of all earnings (wages, salaries, profits, interests payments, rents etc.) in a given period of time (generally a month). Income is a crucial demographic element that is used at a wide variety of customer touch points. Therefore, it is very important to have an income prediction for existing and potential customers. [1]

They also conducted a study of 16 regression algorithms for the task of predicting income in 2015. They found that linear models perform about as well as more complex nonlinear models using a variety of evaluation metrics. [1]

In this paper, we use the Adult Data Set [2], which is a census income dataset from the UCI Machine Learning Repository. It is made up of United States census data from 1994 and has 48,842 examples and 14 features. These features mainly consist of demographic data and are mostly self-explanatory. The fourteen features are *age*, *workclass*, *fnlwgt* (final weight), *education*, *education-num*, *marital-status*, *occupation*, *relationship*, *race*, *sex*, *capital-gain*, *capital-loss*, *hours-per-week*, and *native-country*. The feature *education-num* numerically represents the *education* feature and *fnlwgt* is the final sampling weight given by the census bureau. We propose to solve the binary classification problem of predicting whether or not an individual makes more than $50,000 per year.

We can visualize some characteristics of the data to understand what we're dealing with. Fig. 1 contains a plot of the distribution of the numerical features and Fig. 2 shows their correlations.

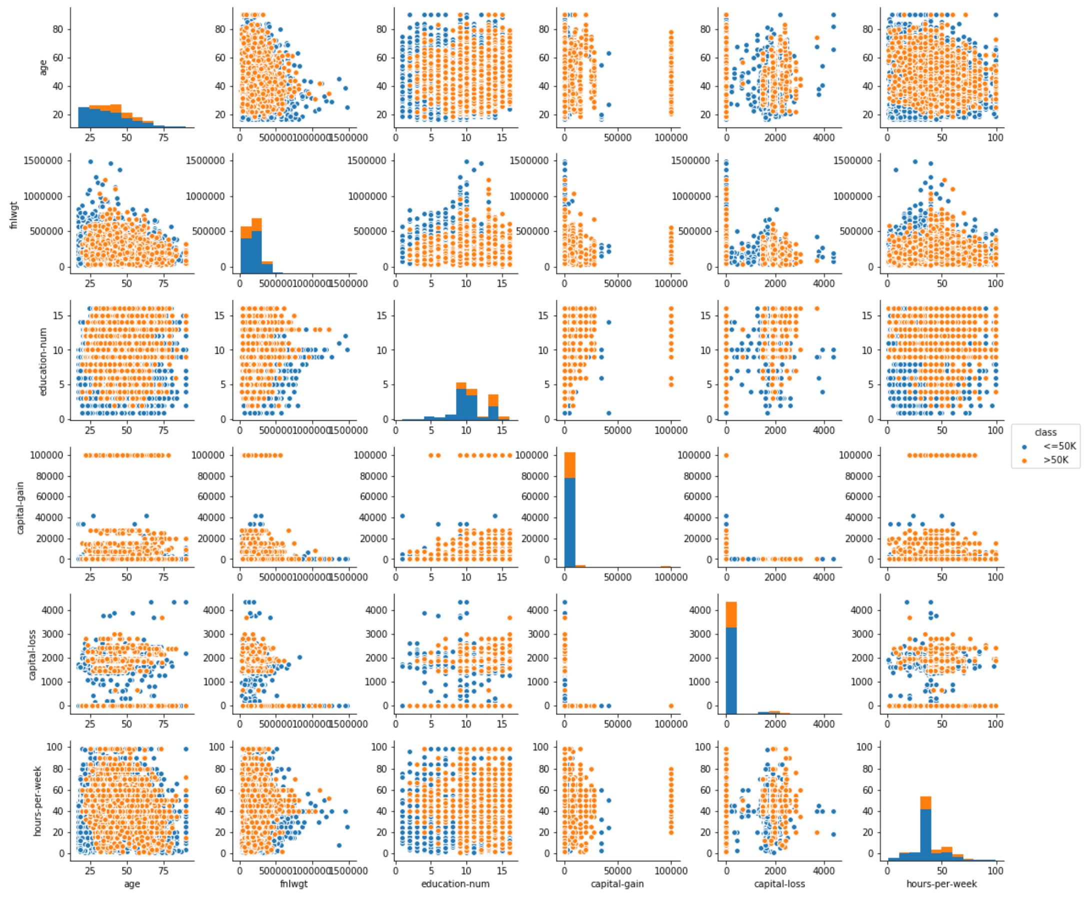

Fig. 1. Distribution of the training data for numerical features.

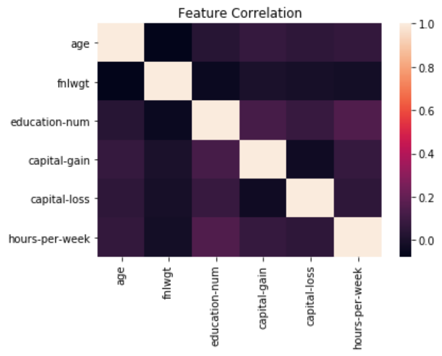

Fig. 2. Correlation of numerical features in the training data.

2 Methods

In order to solve the problem on binary classification, a collection of four different models are evaluated. The accuracy metric is used as a performance measure and every method keeps the training and test time as an efficiency measure. Each model will be discussed in detail below. These models are implemented in the Python library scikit-learn [4]. The data is preprocessed and handled using the

Pandas library, and Seaborn is used for some data visualizations. The data is divided into a training set of 32,561 examples and a test set of 16,281 examples. To fit model parameters and estimate out of sample error, 10-fold cross-validation is used.

2.1 **Linear Classifier:**

We first consider using ridge regression as our linear classifier. Ridge regression combines squared loss with an L2-regularizer. It solves the following problem:

$$min_w = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i + y_i)^2 + \lambda \|w\|^2 \quad [1]$$

The model has the following tradeoffs. The squared loss term is differentiable everywhere, but sensitive to outliers/noise. The L2-regularizer is strictly convex and differentiable, but has dense solutions. Linear models are generally not that flexible (i.e. they can't fit very complex data), but this means it will not overfit the training data. Furthermore, linear models usually have high bias and low variance.
Ridge regression has closed form solution:

$$\hat{w} = \left(XX^T + \lambda I\right)^{-1} Xy^T \quad [2]$$

Training on our roughly 32,000 samples and testing on about 16,000, and by performing 10-fold cross validation ten times, this model achieved an accuracy of 84.0%. It took 0.16s to train and 0.75s to test using 10-fold cross validation prediction. This represents about 0.075s for testing.

2.2 **Gaussian Process (GP) Classifier:**

We then consider using GP classifiers with a radial basis function (RBF) and Matérn kernel. A sample of 1,000 random data points is constructed to train and another 1,000 points to test the models because of memory and runtime issues. This represents about 2% of our data. A Laplace approximation is also employed to approximate the posterior.

As with other Bayesian methods, GPs give a very useful uncertainty measure for each prediction. GPs are a very powerful method that can fit complex hypotheses. However, they do have many negative aspects. It is always the case that one must learn hyperparameters and choose a kernel. They do not scale well with number of training examples because of the $O(n^3)$ time complexity for the matrix inversion. For classification, logistic link function must be used, which is non-Gaussian.

Using 10-times, 10-fold cross validation, we achieved an accuracy score of 73.8% on our GP with the RBF kernel with a length scale of 1, and an accuracy of 74.5% with a Matérn kernel with a length scale of 2 and ν of 3/2.

The GP with the RBF kernel had a p-value of < 0.05 and the Matérn kernel had a p-value of < 0.01.

Though these models were both average and had low p-values, the training and test time was also significant. The RBF GP training time took 1.4s and Matérn train took 1.8s. Comparing this with the Ridge Classifier train time, which was 0.16s, it's clear that the GP training times are heavy. Furthermore, the Ridge classifier was trained on about 32,000 training samples whereas the Gaussian Processes were trained only on 1,000. Even with this disparity, the GP took over eight times (RBF) and ten times (Matérn) longer to train.

2.3 **Dimensionality Reduction:**

We then consider using principal component analysis (PCA) as a dimensionality reduction method. The Adult data set contains categorical features, such as occupation, therefore, dimensionality reduction was not used on it. Instead, we applied PCA to the classic iris flower data set [5], which contains 4 continuous features. Using a ridge classifier, and PCA reduction to 2 components, we achieved, with 10-times 10-fold cross validation, an accuracy of 80.1% and a p-value of less than 0.01. This was down from 83.5% accuracy without PCA.

Using 112 training samples and 38 test samples from the iris data set, it took 0.13s to train the ridge classifier and 0.00056s to test.

With PCA with 2 principal components, the training time was 0.0028s and the test time was 0.00028s. This represents a train time decrease of almost 50-fold and an almost 2-fold decrease in test time. This time reduction would have been particularly useful in training our GPs.

2.4 **Decision Tree Classifier:**

The last model is a decision tree classifier that uses an optimized version of the CART algorithm. Decision trees give interpretable results and can easily handle missing training values. They scale well with number of training data points and don't need to normalize the training data. However, decision trees easily overfit because of how powerful the model is and can give very unstable results.

After creating the classifier and fitting it to the training data, we then computed the scores using 10-fold cross-validation for each fold. The score was determined using the mean accuracy metric. We found the accuracy to be 81%, the training time to be 5.08s, and the prediction time to be 1.821s.

3 Results

The overall results are presented in Fig. 3 below. We used 10-fold cross-validation for each of the models and techniques, and we repeated this 10 times for average results to accurately represent our models.

We performed this on our Ridge Classifier, GPs (using both and RBF kernel and a Matérn kernel), decision tree classifier, as well as a ridge classifier using PCA. We again performed PCA on the Iris data set; we couldn't use it on our data set because it contains categorical features.

The results clearly indicated what we had seen in the past: the Ridge Classifier is the strongest one we've used.

We found an average cross-validation score of 83.95% for the Ridge Classifier, 73.84% for the GP with an RBF kernel, 74.5% for the GP with a Matérn kernel, 81% for the decision tree classifier, and 80.1% success for PCA using 2 principal components (down from 4) on the Iris data set. Without using PCA on the Iris data set, the Ridge Classifier had an average cross-validation score of 83.5%.

Furthermore, we found the p-values for our models using the 10-fold cross-validation. We achieved p-values of < 0.01 for every model except the GP with the RBF kernel for which we achieved a value of < 0.05. Thus, these models strongly appear to be valid and accurate.

The Ridge Classifier is thus clearly the strongest model of the ones we've used. If we were able to use PCA with it, PCA might also prove useful in reducing complexity and learning time without compromising on the success rate.

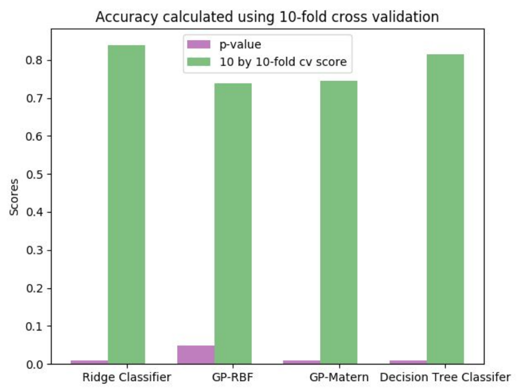

Fig. 3. Plot of each model's p-value and accuracy.

4 Lessons Learned

We found the process of systematically constructing and evaluating a collection of machine learning models and techniques to be a very informative experience. We learned that it is always a good idea to try simple models first (e.g. linear) because they may have the best predictive accuracy and are the easiest to implement. Our data set is too big to use GPs, but we could have tried sparse GPs. We also learned how hard is to implement categorical PCA in Python. PCA also showed us the tradeoff in efficiency vs performance. It seems like PCA may sometimes decrease performance, but increase efficiency. Additionally, being precise in model comparison is critical. Maintaining a constant training and test set is very important to ensure comparable models and an accurate estimate out of sample performance. Lastly, decision trees offer a nice way of interpreting results, but this is debatable.

There are many more models we would like to try on our data set. We suspect that a neural network or random forest would have been more effective.

References

[1] Azamat Kibekbaev and Ekrem Duman. 2015. Benchmarking Regression Algorithms for Income Prediction Modeling. 2015 International Conference on Computational Science and Computational Intelligence (CSCI) (2015). DOI:http://dx.doi.org/10.1109/csci.2015.162
[2] Ronny Kohavi and Barry Becker. 1996. UCI Machine Learning Repository: Adult Data Set. (May 1996). Retrieved May 3, 2018 from https://archive.ics.uci.edu/ml/datasets/adult
[3] Elan Miller and Louis Schlessinger. 2018. Project Wiki. (May 2018). Retrieved May 3, 2018 from https://github.com/elanbmiller/MLApplicationProject/wiki
[4] Pedregosa, F., Varoquaux, G., Gramfort, A. & Michel, V. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
[5] Ronald Fisher. 1988. UCI Machine Learning Repository: Iris Data Set. (July 1988). Retrieved May 7, 2018 from https://archive.ics.uci.edu/ml/datasets/iris