Supervised Learning - Report

Xicheng Huang (xhuang343) - Fall 2018

Table of Content

Introduction

For this assignment, I picked two completely different datasets in terms of their characteristics for classification problems so that I can learn applying supervised learning algorithms on a more well-rounded way. The first dataset is somewhat a larger dataset where it consists of more than 48000 records. It is a binary classification problem and the features are categorical. The second dataset I picked is a much smaller dataset with only 304 records, and it is also a multi-class problem. For each dataset, I will explain what I did to preprocess them and then go into each of the supervised learning algorithms I applied and their corresponding performance.

For this assignment, I primarily used Python's scikit-learn library to perform machine learning tasks and used various libraries, including matplotlib and sklearn-evaluation to graph various performance metrics and results of the applied algorithms.

Dataset One

Preprocessing

The first dataset is from UC Irvine Machine Learning Repository, and it is called Adult data set. This is an extraction from the 1994 Census database and prediction task is to determine whether a person makes over 50K a year. Once I loaded the data with headers and the right delimiter, without any preprocessing, the dataset looks like this:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	incomeCategory
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

As you can seem the dataset is full of categorical features and the label (incomeCategory) is a string representation as well. Because scikit-learn's classification algorithms will not accept any string representation features, one of the first tasks is to convert them to numerical. I could enumerate the categories for each feature, for example, 0 for "Bachelors" and 1 for "HS-grad" in education, but scikit-learn will treat them as continuous data, as a result, the models could provide wrong "understanding" of these features. So the solution is one-hot-encode all of them. After encoding, it will add extra columns for each category of each of the feature. For example, row 0 will have education_Bachelors as 1, meaning it indeed has a education of

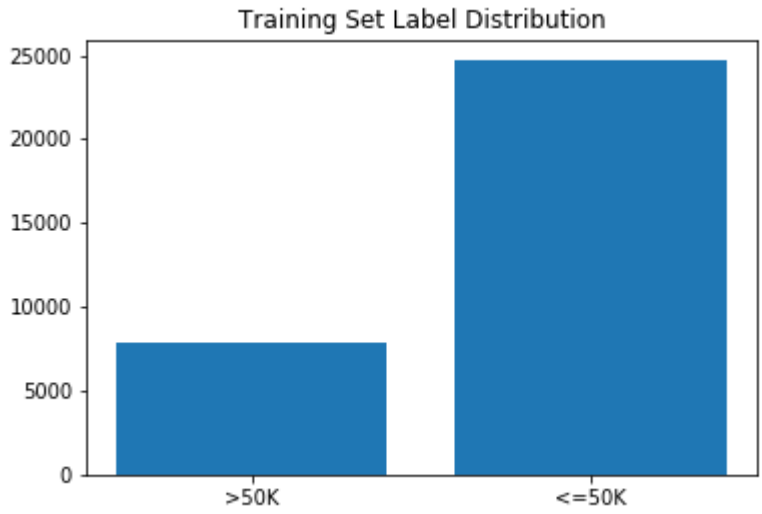
Bachelors for the feature, and everything other columns for education is 0. Also, I converted the labels into 0s and 1s where 0 is "<=50K" and 1 is ">50K". Then, there were also a handful missing data, so I just set them as 0. After preprocessing, the dataset looks like this:

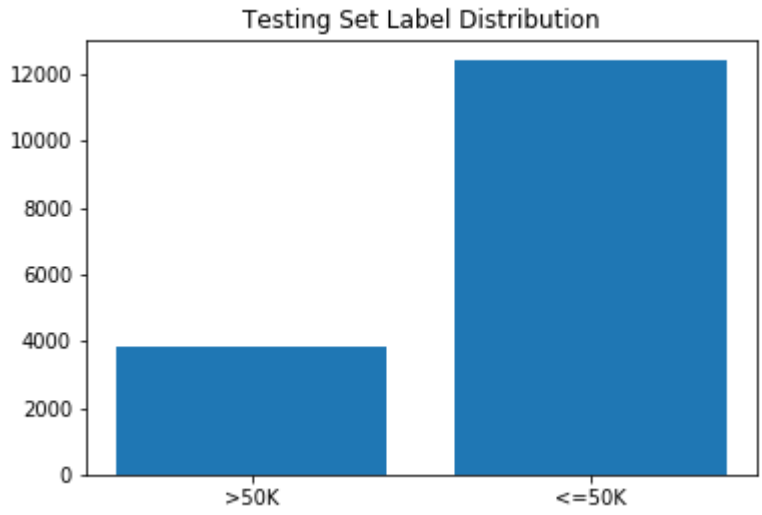
	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	workclass_?	workclass_Federal-gov	workclass_Local-gov	workclass_Never-worked	...	native-country_Puerto-Rico	nat country_Scotl
0	25	226802	7	0	0	40	0	0	0	0	...	0	
1	38	89814	9	0	0	50	0	0	0	0	...	0	
2	28	336951	12	0	0	40	0	0	1	0	...	0	
3	44	160323	10	7688	0	40	0	0	0	0	...	0	
4	18	103497	10	0	0	30	1	0	0	0	...	0	

5 rows × 108 columns

Data Investigation

After preprocessing them, I randomly divided the dataset up into thirds, 2/3 will be used for training, and 1/3 will be used to testing. Then, I wanted to check the labels to see if the distribution of classes are even. Here is the result:

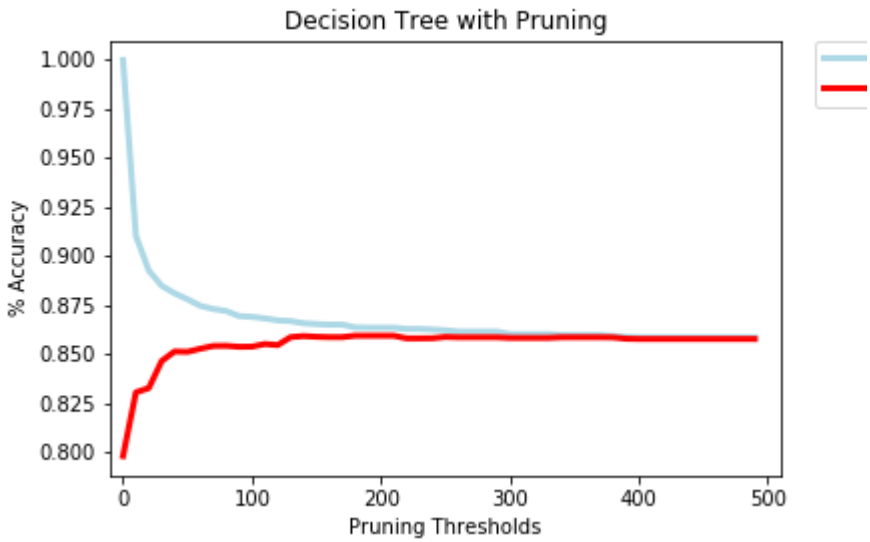




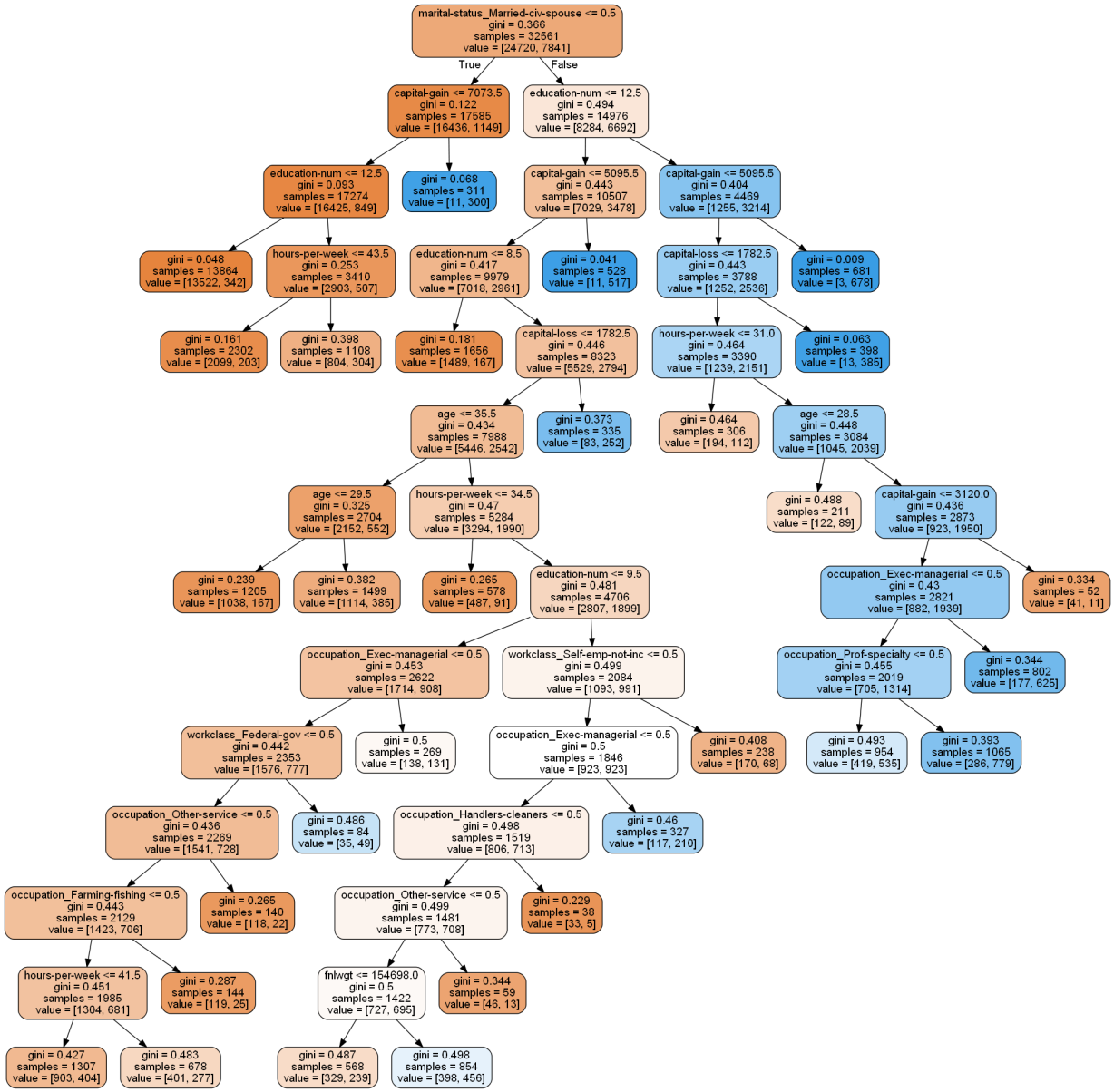
As you can see, there are far more "<=50K" labels in both the training set and testing set. So to make sure the machine learning models are performant, I should not only look at accuracy score but also f1 scores.

Decision Tree

For decision tree classifier, I used scikit-learn's `DecisionTreeClassifier()` to create the model. Without pruning, the tree has more than 9000 nodes! This is very large and definitely susceptible to overfitting. So I used a custom pruning method to prune the tree after modeling. The custom pruning method takes in a threshold parameter and utilize it to check the value of each node and if the smallest value of the node is below that threshold, it will prune it and its children out. Here is a graph representing the benefit of this pruning process:



As you can see, without any pruning, the model is overfitting there it is at almost 100% accuracy when predicting training set while it is not doing so hot for testing set. As the pruning threshold increases, predictions for training set and testing set are converaging, and overfitting fades out. The final tree looks like this:



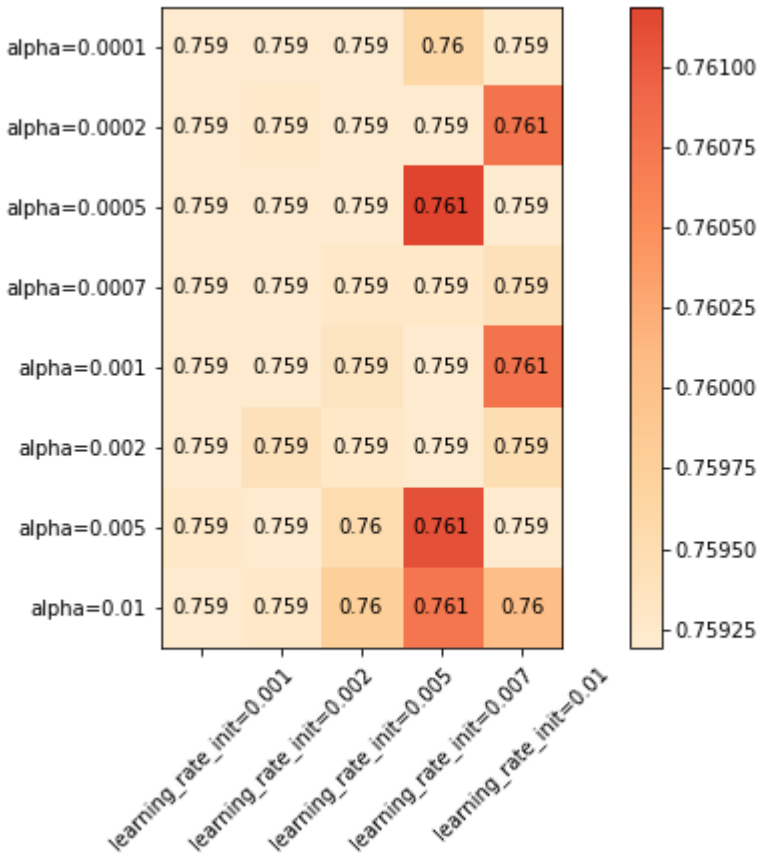
Using a model with pruning threshold of 350, I created a f1 score report.

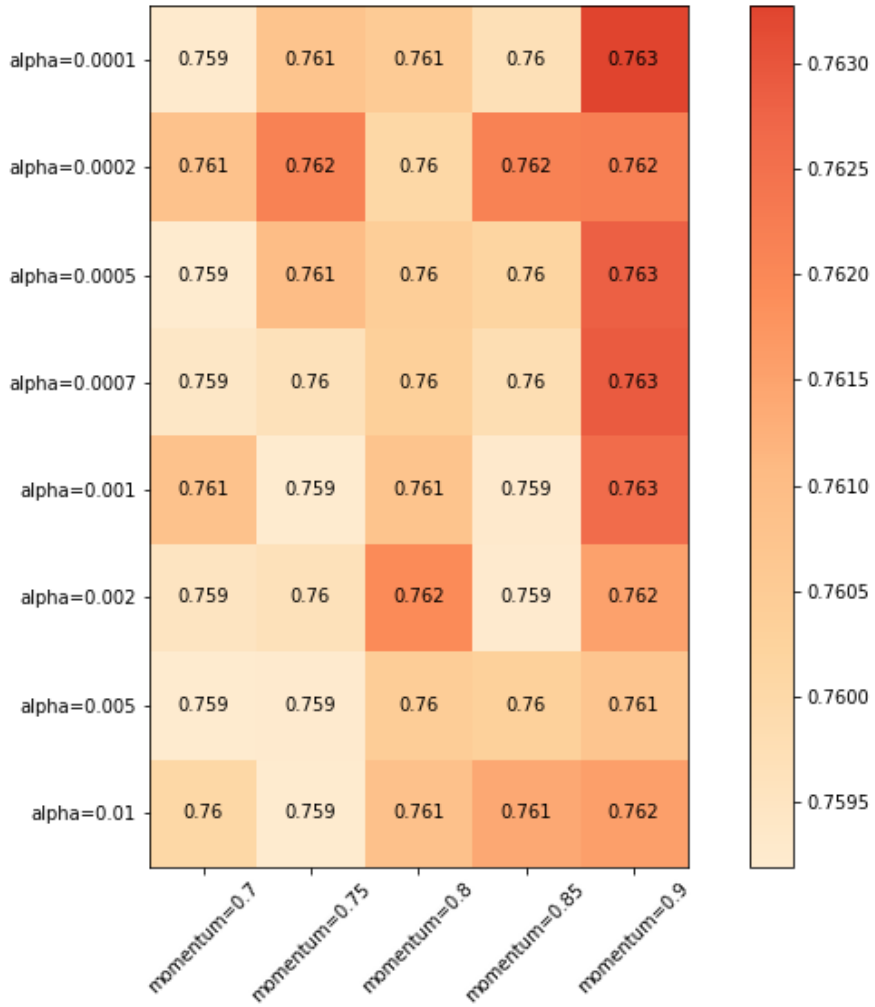
	precision	recall	f1-score	support
<=50K	0.88	0.94	0.91	12435
>50K	0.75	0.60	0.67	3846
avg / total	0.85	0.86	0.85	16281

It is pretty good for determining "<=50K" class but not so good at ">50K". Let's look at the what the next algorithm can do.

Neural Networks

For this algorithm, I used `MLPClassifier()`, and specifically, I used logistic regression in combination of stochastic gradient descent for activation because according to the lecture, calculus is better 😃. I also incorporated cross-validation and hyperparameter tuning to try to get the best result from this algorithm. For this, I used `GridSearchCV()` provided by scikit-learn. The result is this:


alpha=0.0001
alpha=0.0002
alpha=0.0005
alpha=0.0007
alpha=0.001
alpha=0.002
alpha=0.005
alpha=0.01
0.759 0.759 0.759 0.76 0.759
0.759 0.759 0.759 0.759 0.761
0.759 0.759 0.759 0.761 0.759
0.759 0.759 0.759 0.759 0.759
0.759 0.759 0.759 0.759 0.761
0.759 0.759 0.759 0.759 0.759
0.759 0.759 0.76 0.761 0.759
0.759 0.759 0.76 0.761 0.76
0.76100
0.76075
0.76050
0.76025
0.76000
0.75975
0.75950
0.75925
learning_rate_init=0.001
learning_rate_init=0.002
learning_rate_init=0.005
learning_rate_init=0.007
learning_rate_init=0.01


alpha=0.0001
alpha=0.0002
alpha=0.0005
alpha=0.0007
alpha=0.001
alpha=0.002
alpha=0.005
alpha=0.01
0.759 0.761 0.761 0.76 0.763
0.761 0.762 0.76 0.762 0.762
0.759 0.761 0.76 0.76 0.763
0.759 0.76 0.76 0.76 0.763
0.761 0.759 0.761 0.759 0.763
0.759 0.76 0.762 0.759 0.762
0.759 0.759 0.76 0.76 0.761
0.76 0.759 0.761 0.761 0.762
0.7630
0.7625
0.7620
0.7615
0.7610
0.7605
0.7600
0.7595
momentum=0.7
momentum=0.75
momentum=0.8
momentum=0.85
momentum=0.9