Supervised Learning - CS7641 Fall 2021

Huifu Tian

September 10, 2021

In this assignment, I will use the Polish companies bankruptcy data set and the MNIST handwritten digits data set to compare various supervised learning algorithms. All of the analysis was done using the scikit-learn library compatible with Python 3.

1 Polish Data

1.1 Data Overview

This dataset is a collection of Polish companies spanning 2007-2013. The dataset contains a total of roughly 700 bankrupt companies corresponding to 2400 financial reports in the period between 2007 and 2013. The data is separated into five datasets - 1stYear, 2ndYear, 3rdYear, 4thYear, 5thYear. Each row in every data set contains the financial report of one company in a single year.

- 1stYear: 7027 surveyed companies in 2007, 271 of which went bankrupt by the start of 2013
- 2ndYear: 10173 surveyed companies in 2008, 400 of which went bankrupt by the start of 2013
- 3rdYear: 10503 surveyed companies in 2009, 495 of which went bankrupt by the start of 2013
- 4thYear: 9792 surveyed companies in 2010, 515 of which went bankrupt by the start of 2013
- 5thYear: 5910 surveyed companies in 2011, 410 of which went bankrupt by the start of 2013

The attributes are are shown in Table 1.

The output consists of one column where 1 means bankruptcy and 0 means non-bankruptcy.

add more stats exploring the data such as autocorrelations, class sizes, etc

1.2 Preprocessing

Minimal preprocessing was done to keep the focus on analysis. I combined all of the data sets to form a new dataset of over 43000 rows, which may introduce oversampling bias since it's not guaranteed that each company that is marked with a bankruptcy status=1 only shows up once across the 5 data sets. is it called oversampling bias? All of the infrequent NaNs in the attribute columns are replaced with 0's. (There are no NaNs in the output column.) Finally, I divide the data into 70% train and 30% test sets while applying stratification to ensure uniform distribution of classes between the two groups.

1.3 Decision Tree

Baseline model

The baseline model uses sklearn's DecisionTreeClassifier class and is trained with unlimited tree depth and the Gini measure of impurity. The error measure that we care about depends largely on the reason we are doing this study and whether or not our data is imbalanced. If the goal is to predict with high confidence that a company predicted to go bankrupt will actually go bankrupt (i.e. to construct a portfolio of assets to short), then we might care a lot about precision. On the other hand, if we care about finding all the companies that might go bankrupt, then we should use recall, or balanced accuracy, which is the mean of recall on both classes. The other observation is that our data is imbalanced: about 4.5% of the data falls in the minority (bankrupt) class. Table 2 shows the mean metric scores from fitting the classifier on the training data and scoring on the test data with various undersampling ratios across 20 iterations of random sampling from the majority class.

Before tuning the model, it's worth noting that the decision tree classifier exhibits the general behavior one tends to see with imbalanced data sets, namely that the classifier prefers to minimize error for the majority class at the expense of the minority class. As the sample rebalancing approaches parity between both classes (shown by the progression from 3:1 to 1:1 ratio), precision, recall, and accuracy of the minority class are all improved. As a bonus, fitted tree depth also becomes smaller, which is preferred because we like simpler models over more complex models.

Model Tuning

Impurity

Table 3 shows the same analysis as Table 2 using the entropy instead of gini as the impurity measure. The reuslts are similar. Since the entropy measure takes longer to calculate due to logs in the measure and it doesn't offer significant improvements, we can stick to using gini measure for the rest of the analysis.

Table 1: Polish companies bankruptcy data attributes

X1 net profit / total assets	X33 operating expenses / short-term liabilities
X2 total liabilities / total assets	X34 operating expenses / total liabilities
X3 working capital / total assets	X35 profit on sales / total assets
X4 current assets / short-term liabilities	X36 total sales / total assets
X5 [(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365	X37 (current assets - inventories) / long-term liabilities
X6 retained earnings / total assets	X38 constant capital / total assets
X7 EBIT / total assets	X39 profit on sales / sales
X8 book value of equity / total liabilities	X40 (current assets - inventory - receivables) / short-term liabilities
X9 sales / total assets	X41 total liabilities / ((profit on operating activities + depreciation) × (12/365))
X10 equity / total assets	X42 profit on operating activities / sales
X11 (gross profit + extraordinary items + financial expenses) / total assets	X43 rotation receivables + inventory turnover in days
X12 gross profit / short-term liabilities	X44 (receivables * 365) / sales
X13 (gross profit + depreciation) / sales	X45 net profit / inventory
X14 (gross profit + interest) / total assets	X46 (current assets - inventory) / short-term liabilities
X15 (total liabilities * 365) / (gross profit + depreciation)	X47 (inventory * 365) / cost of products sold
X16 (gross profit + depreciation) / total liabilities	X48 EBITDA / total assets
X17 total assets / total liabilities	X49 EBITDA / sales
X18 gross profit / total assets	X50 current assets / total liabilities
X19 gross profit / sales	X51 short-term liabilities / total assets
X20 (inventory * 365) / sales X21 sales (n) / sales (n-1)	X52 (short-term liabilities * 365) / cost of products sold) X53 equity / fixed assets
X22 profit on operating activities / total assets	X54 constant capital / fixed assets
X23 net profit / sales	X55 working capital
X24 gross profit (in 3 years) / total assets	X56 (sales - cost of products sold) / sales
X25 (equity - share capital) / total assets	X57 (current assets - inventory - short-term liabilities) / (sales - gross profit - depreciation)
X26 (net profit + depreciation) / total liabilities	X58 total costs /total sales
X27 profit on operating activities / financial expenses	X59 long-term liabilities / equity
X28 working capital / fixed assets	X60 sales / inventory
X29 logarithm of total assets	X61 sales / receivables
X30 (total liabilities - cash) / sales	X62 (short-term liabilities *365) / sales
X31 (gross profit + interest) / sales	X63 sales / short-term liabilities
X32 (current liabilities * 365) / cost of products sold	X64 sales / fixed assets

Table 2: Mean Decision Tree Classifier Baseline Scores with Undersampling

Metric	No Undersampling[1]	3:1 [2]	2:1	1:1
Accuracy	0.95	0.83	0.81	0.78
F1	0.48	0.67	0.71	0.78
Precision	0.46	0.66	0.71	0.78
Recall	0.50	0.68	0.72	0.79
Balanced accuracy	0.73	0.78	0.78	0.78
Fitted tree depth	33	24.25	22.8	19.65

Table 3: Mean Decision Tree Classifier Baseline Scores with Undersampling using Entropy

Metric	No Undersampling[1]	3:1 [2]	2:1	1:1
Accuracy	0.95	0.84	0.81	0.79
F1	0.52	0.68	0.71	0.79
Precision	0.51	0.68	0.71	0.78
Recall	0.50	0.69	0.72	0.79
Balanced accuracy	0.75	0.79	0.79	0.79
Fitted tree depth	26	21.8	20.9	19.75

Figure 1: Validation Curve: Max Tree Depth

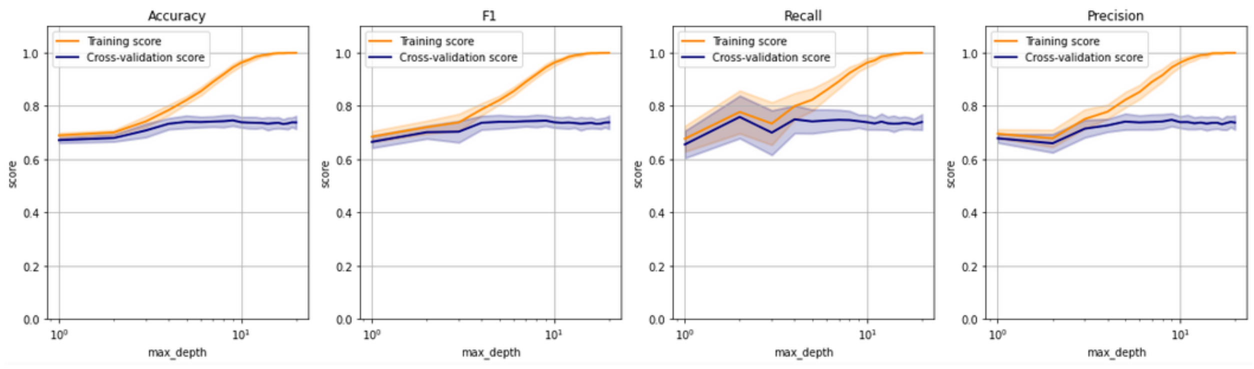
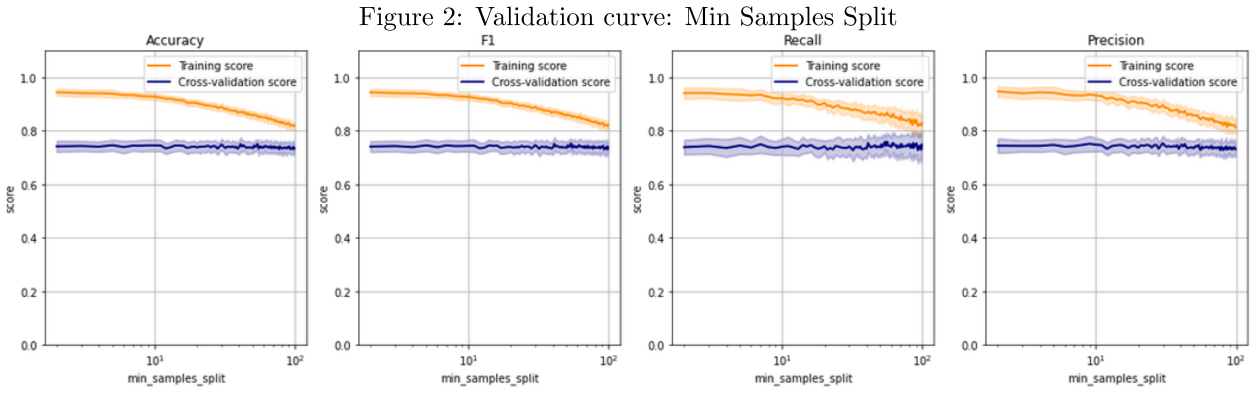


Figure 2: Validation curve: Min Samples Split



Max Tree Depth

Without bounding max tree depth, the baseline model will fit the data to a tree with average depth of 19.65 (Table 2) using 1:1 rebalancing across 20 random samplings across the majority class. To determine the ideal max tree depth, I use a modified version of the Stratified K Folds validation algorithm. Rather than selecting k folds to split one set of the post-undersampled training data into a train and validation set, I select 2 folds across k sets of training data. The reason for this modification is that I'm undersampling the majority class. To reduce the variance of results across multiple trials as much as possible, hence k folds, it makes more sense to randomly pick from a much bigger pool of data than to undersample first then pick from a smaller pool of data. The results are shown in Figure 1.

The validation curves in Figure 1 show increasing training scores in general. The reason is increasing the max tree depth allows the model to fit the training data to increasing accuracy. With a tree of around 20 nodes deep, the model is able to almost perfectly fit the training data. At that point, one would expect generalization error to start decreasing. What's shown in figure 1 is that most of the increase in the various score metrics peak out around max depth of 9. While the training score continues to improve, we prefer to cap max tree depth to the smallest number that maximizes the testing scores. This decision follows from the principle of Occam's Razor which states that simpler models are preferred over more complex ones.

Table 4: Optimal Max Tree Depth

Metric	Max Tree Depth	Min Samples Split	Min Samples Leaf	haha
Accuracy	0.95	0.84	0.81	0.79
F1	0.52	0.68	0.71	0.79
Precision	0.51	0.68	0.71	0.78
Recall	0.50	0.69	0.72	0.79
Balanced accuracy	0.75	0.79	0.79	0.79
Fitted tree depth	26	21.8	20.9	19.75

Min Samples Split

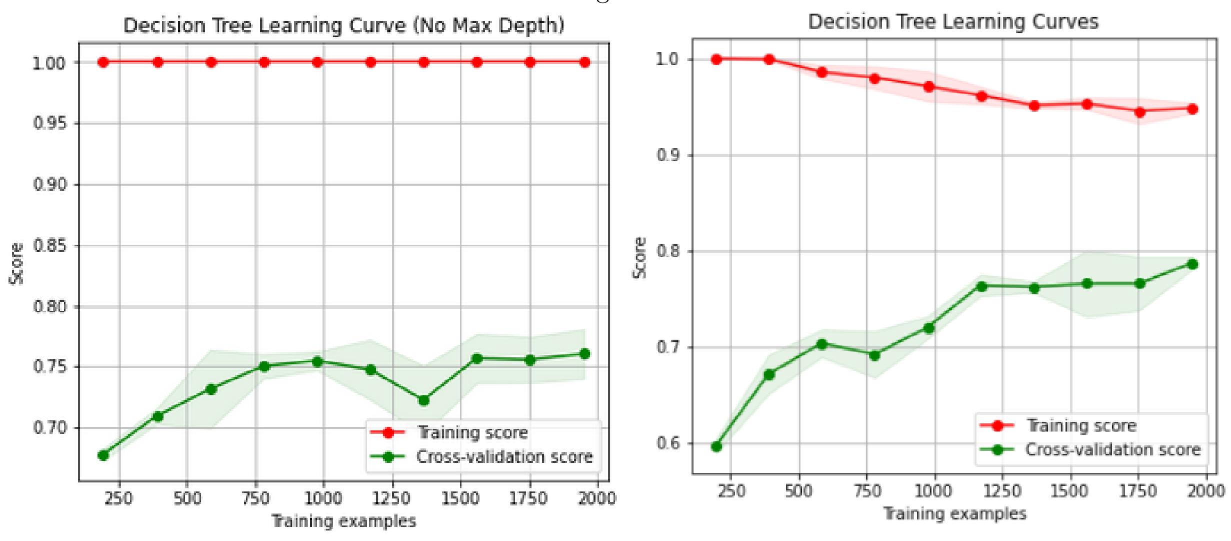
Another hyperparameter we can use to tune the model is the minimum number of samples required for a node before the node can be further split. I repeat the same procedure with the modified Stratified K Folds with and without controlling for max tree depth. The validation curves for minimum samples split while setting max tree depth to the optimal value of 9 are shown in Figure 2. Both sets of validation curves (setting and not setting max depth - not shown) look similar. Metric scores on the training set are decreasing from 2 to 100 are decreasing while the scores on the validation set are flat. It's possible to find an "optimal" n such that test scores are maximized but additional studies must be done to validate that the maximum value at the "optimal" n is statistically different from another value. do this analysis?

Cost Complexity Pruning

The last parameter I will tune is the cost complexity pruning alpha. This parameter reduces the weak links in the tree by removing those nodes and all of their child trees. The alphas are a regularization parameter that penalizes the number of leaves in the fitted tree. I run cost complexity pruning after setting max_depth

[1]Only one iteration of fitting
[2]ratio of the post-undersampled majority to minority class

Figure 3



to the optimal number. The result (not shown show if there's space) indicate that doing post pruning does not improve accuracy of model when evaluated on the test set.

Learning Curves

To understand whether our model is overfitting or underfitting, I make two learning curves: one without restrictions, one with max tree depth. In the case without setting max_depth, the training accuracy scores stay at roughly 100%. This shows that the model was likely too complex and fit the training model perfectly. The idea that this model is overfitting the data is also supported by the fact that the validation scores remain flat, meaning that the model is unable to generalize. Once I introduce max_depth, the training model's accuracy scores start to decrease as we limit the model's complexity. The result is that a simpler model does better at generalization, which is shown by the much improved scores on the test data. There is still a gap between the training curve and the validation curve at the 2000th sample, which means that further limiting of the model's complexity may improve the validation score. Also, validation scores are still in the middle of improving at the 2000th sample, which implies that having more samples would improve the model's ability to fit the data. discuss variance and bias

Final model and performance

The model with max_depth set to 9 yields is run 20 times using 1:1 balanced sampling between the two classes, each time picking a 70% training and 30% testing set. The mean accuracy across the 20 iterations is 0.79. Show in final summary table

1.4 Neural Network

Baseline model

Similar to the decision trees, I use the full data set which combines all five data sets and use a 70-30 train-test split. Table 5 column 1 shows the result of fitting on the full 70% training set. The results are iterated over 10 sets. Not surprisingly, the model yields an accuracy of 94% which is roughly equivalent of always guessing label of 0. Columns 2 and 3 explore the effect of undersampling the majority class on the various score metrics. As we move from the full data set to a 1:1 rebalanced data set, accuracy decreases while f1, precision, recall, and balanced accuracy all increase. It can be seen that neural networks suffer from the same majority class bias as decision trees: accuracy is maximized by focusing on fitting the model to the majority class while neglecting the minority class. Intuitively, that makes sense, the algorithm strives to maximize accuracy. High accuracy can only be achieved by catering more attention to the majority class. Column 4 normalizes all of the features but I ran into nonconvergeance at 1000 iterations. I increase the number of iterations to 2000 and I can converge again. The performance with normalized features jumps by 50% across all of the metrics. explain why.

Model Tuning

Size of Hidden Layers

The training and cross validation accuracy scores by varying the number of nodes in a one-hidden layer net are shown in Figure 3. The parameters and results correspond to column 4 in Table 5. The accuracy curve on the train set starts low but increases as more complexity is enabled on the model. Also at first, the increasing complexity in the model benefits the validation scores. The question is what's the optimal number of nodes in the hidden layer that maximizes generalization accuracy? And the answer is 42. Any further increases in model complexity only marginally increases the model fit on the training data while gradually lowering the ability to generalize on the validation set. This is overfitting. One can also see from the training scores after n=42 start to display higher variance, which means that the model's accuracy becomes more dependent on particular idiosyncrasies in the weights and the data itself. check last sentence.

At this point, I can do a grid search to find the optimal number of nodes on a 2-hidden layer network. But before that, I will hold the number of nodes at 42 on the first layer and repeat the same procedure as before

Figure 3: Hidden Layer 1 Size

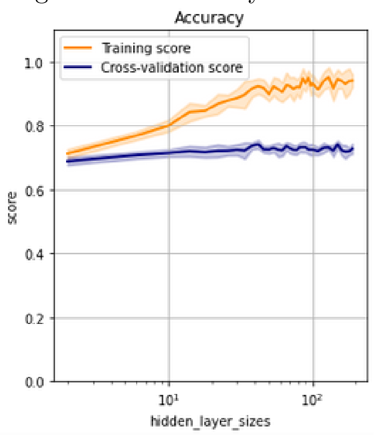


Figure 4: Hidden Layer 2 Size

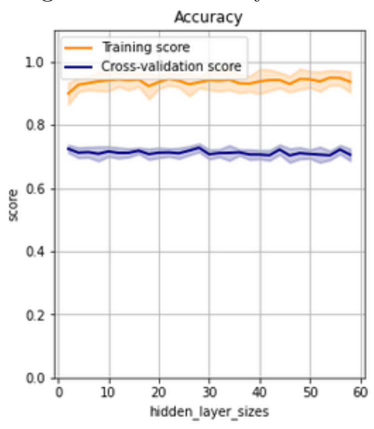


Figure 5: Regularization Alpha

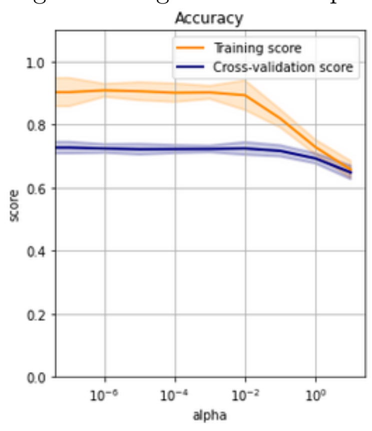


Table 5: Neural Network Parameters and Results

Parameter	1	2	3	4	5	6
solver	adam	adam	adam	adam	adam	adam
activation	relu	relu	relu	relu	relu	relu
alpha	1e-4	1e-4	1e-4	1e-4	1e-4	1e-7
hidden_layer_sizes	(30,)	(30,)	(30,)	(30,)	(42,)	(42,)
max_iter	1000	1000	1000	2000	3000	3000
learning_rate	constant	constant	constant	constant	constant	constant
learning_rate_init	0.001	0.001	0.001	0.001	0.001	0.001
momentum	0.9	0.9	0.9	0.9	0.9	0.9
Normalized?	No	No	No	Yes	Yes	Yes
Undersampled?	No	3:1	1:1	1:1	1:1	1:1
Results						
Accuracy	0.94	0.66	0.57	0.74	0.75	
F1	0.04	0.36	0.57	0.75	0.75	
Precision	0.11	0.37	0.58	0.74	0.75	
Recall	0.03	0.37	0.58	0.75	0.76	
Balanced accuracy	0.51	0.56	0.58	0.74	0.75	

on the second layer. The results are shown in Figure 4. As you can see, the accuracy on the cross-validation curve is flat, which shows that the extra layer does not seem to add further benefits to the 1-layer model.

Regularization Alpha

Another parameter to vary is the L2 regularization alpha. This parameter is the scalar term to an added term in the cost function to be minimized. The penalty term is the sum of the weights squared multiplied by alpha/2, which means that any model that minimizes this function will also need to make sure the weights are small as well. The larger the alpha, the greater the penalty and more we can limit potential overfitting. The results are presented in Figure 5.

1.5 KNN

Baseline Model

1.6 Boosting

Baseline Model

1.7 SVM

Baseline Model

Table 5: KNN Parameters and Results

Parameter	1	2	3	4
n_neighbors	5	5	5	sgd
weights	uniform	uniform	uniform	relu
algorithm	auto	auto	auto	0.01
p	2	2	2	(30,)
Undersampled?	No	3:1	1:1	
Results				
Accuracy	0.95	0.74	0.65	0.643
F1	0.02	0.38	0.65	
Precision	0.10	0.47	0.65	0.648
Recall	0.01	0.31	0.65	
Balanced accuracy	0.50	0.60	0.65	0.643

Table 5: Boosting Parameters and Results

Parameter	1	2	3	4
base_estimator	DecisionTreeClassifier		sgd	sgd
n_estimators	100	100	relu	relu
learning_rate	1	1	0.01	0.01
algorithm	SAMME.R	SAMME.R	(30,)	(30,)
Normalized?	No	No	Yes	Yes
Undersampled?	No	1:1		1:1
Results				
Accuracy	0.95	0.80	0.757	0.643
F1	0.23	0.80	0.151	0.635
Precision	0.57	0.80	0.597	0.648
Recall	0.14	0.81	0.089	0.624
Balanced accuracy	0.57	0.80	0.535	0.643

Table 5: KNN Parameters and Results

Parameter	1	2	3	4
solver	sgd	sgd	sgd	sgd
activation	relu	relu	relu	relu
alpha	0.01	0.01	0.01	0.01
hidden_layer_sizes	(30,)	(30,)	(30,)	(30,)
max_iter	1000	1000	1000	1000
learning_rate	constant	constant	constant	constant
learning_rate_init	0.001	0.001	0.001	0.001
momentum	0.9	0.9	0.9	0.9
Normalized?	No	Yes	Yes	Yes
Undersampled?	No	No	3:1	1:1
Results				
Accuracy		0.952	0.757	0.643
F1		0.003	0.151	0.635
Precision		0.333	0.597	0.648
Recall		0.002	0.089	0.624
Balanced accuracy		0.501	0.535	0.643