Setting up data for modeling

Like any other Python code along, we need to load the packages first:

# Load up packages import pandas as pd import numpy as np import matplotlib.pyplot as plt %matplotlib inline from sklearn.linear\_model import LogisticRegression from xgboost import XGBClassifier from sklearn.model\_selection import train\_test\_split from sklearn.metrics import roc\_curve, auc, roc\_auc\_score import h2o from h2o.automl import H2OAutoML

Next, we’ll load the data called the variable, data:

# Load data data = pd.read\_csv('bankruptcy\_Train.csv')

Personally, I know this data set is pretty clean, but we’ll go through some of the motions:

# Seeing how big the dataset is data.shape

(10000, 65)

The above shows up 10,000 rows with 65 columns. A table form would be seen here:

# Glancing at what some of the values are data.head()

0\*AirsbL0XvtiPJ9Pe?q=20

Everything looks covered into a normalized form for easy machine learning predictions, but let’s check for any null values:

# Checking for null values data.isnull().values.any()

False

Excellent, no Nan values to worry about. Our desired output for bankruptcies in this dataset is called “class.” I don’t like it, so I’m going to rename it to “target”, as seen below:

data.rename(columns={'class':'target'}, inplace=True)

Now, we can split the features and outputs for our models:

# For features in h2o model cont\_names = data.columns[:-1] ﻿ ﻿﻿#Setting up desired output and features for logistic regression and xgboost models output = data[data.columns[-1]] features = data[data.columns[:(data.shape[1]-1)]]

Out of curiosity, let’s see how imbalanced the data set is.

#Check the balance of outputs output.value\_counts()

0\*EmIppDwr2QqwPN95?q=20

0 means non-bankrupt companies, while 1 means bankrupt companies. No surprise here that there aren’t that many bankruptcies. This leads to some tricky situations like how to predict with a data imbalance for logistic regression, but it doesn’t effect it much with xgboost due to how that algorithm works. Luckily for logistic regression, the model has a parameter to adjust for class imbalances as we will soon see!

Predicting bankruptcies with logistic regression and xgboost

After all that data set up, we can finally start building the models for bankruptcy prediction. In predictive models, it is standard practice to split the data into a training and test set. The model will learn from the training set, and we will see how it learned with the test set.

#splits data into X (features) and y (predictions) X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, output, test\_size=0.2, random\_state=42) train = pd.concat([X\_train, y\_train], 1) test = pd.concat([X\_test, y\_test], 1)

Since we want to look at two models (logistic and xgboost), I set up the code below to have both of the models run and go into the same receiver operating characteristic (ROC) curve.

plt.figure()# Add the models to the list that you want to view on the ROC plot models = [ { 'label': 'Logistic Regression', 'model': LogisticRegression(class\_weight='balanced'), }, { 'label': 'XGBoost Classifier', 'model': XGBClassifier(max\_depth=10, n\_estimators=300), } ]# Below for loop iterates through your models list for m in models: model = m['model'] # select the model model.fit(X\_train, y\_train) # train the model y\_pred=model.predict(X\_test) # predict the test data # Compute False postive rate, and True positive rate fpr, tpr, thresholds = roc\_curve(y\_test, model.predict\_proba(X\_test)[:,1]) # Calculate Area under the curve to display on the plot auc = roc\_auc\_score(y\_test,model.predict(X\_test)) # Now, plot the computed values plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (m['label'], auc)) # Custom settings for the plot plt.plot([0, 1], [0, 1],'r--') plt.xlim([0.0, 1.0]) plt.ylim([0.0, 1.05]) plt.xlabel('1-Specificity (False Positive Rate)') plt.ylabel('Sensitivity (True Positive Rate)') plt.title('Logistic Regression v. XGBoost ROC') plt.legend(loc="lower right") plt.show()

For ROC curves, the higher number is the better model. Think about the ROC curve like a better form of gauging accuracy for classification problems. It plots both the true positive (actually predicted to be the target variable) and false positive (predicted but not the actual target variable) rate into one model performance statistic.

0\*9RZNq9AR8RBDbHUK?q=20

In our example, the logistic regression was better than the xgboost with those given settings. But given the history of xgboost winning like almost every Kaggle competition, we know xgboost can do better. It also just happens to be an easier way for us too for model tuning.

Predicting bankruptcies with h2o

This is where h2o gets fun — increased automation for our machine learning models!

# starting up h20 h2o.init()

0\*ldZ41w-cLoKvizim?q=20

You’ll see something like the list above, except longer. It just means your h2o stuff works. Some of you might need to install Java, but h2o will tell you. Now we can begin training (get a cup of tea, while it trains for an hour):

# Training phase set up data = h2o.H2OFrame(train) ﻿ ﻿﻿# Setting up features and output for h2o models data['target'] = data['target'].asfactor() y = "target" cont\_names = cont\_names.tolist() x = cont\_names ﻿ ﻿﻿# Setting up max time of model training aml = H2OAutoML(max\_runtime\_secs= 3600, max\_models=60, sort\_metric='AUC') aml.train(x = x, y = y, training\_frame = data) ﻿ ﻿﻿# Displaying best models built lb = aml.leaderboard lb.head(rows=lb.nrows)

0\*yD8FJhNFRPeOIl4T?q=20

The table above shows us the top 5 models h2o built for us. The top 4 shows all gradient boosted models (xgboost falls under that type of model), and the fifth best model is a stacked ensemble. A stacked ensemble is basically a bunch of machine learning models made to predict together. In contrast our GBM model is just a single type of machine learning model.

Anyways, let’s make some predictions with our best GBM model and see the ROC score.

# Creating Predictions of best model hf = h2o.H2OFrame(test) preds = aml.predict(hf) preds = preds.as\_data\_frame() preds['p\_p0'] = np.exp(preds['p0']) preds['p\_p1'] = np.exp(preds['p1']) preds['sm'] = preds['p\_p1'] / (preds['p\_p0'] + preds['p\_p1']) ﻿ ﻿﻿# ROC score of best model roc\_auc\_score(y\_test, preds['sm'])

0.887

Well, a ROC score of 0.89 beats our above logistic model of 0.78 — yay! For those of you curious on the h2o model’s settings, we can see it like this:

# Settings of best model aml.leader.summary()

0\*YFkTm5BoFJbKevd8?q=20

The cool part about this is if your work does not allow the h2o package but allows the underlying model, then you can just copy the settings over! Another thing to note is you can always save your model and load it up later to save training time.

# Saving model for future use h2o.save\_model(aml.leader, path = "/model\_bankrupt", force=True)

‘C:\\model\_bankrupt\\GBM\_grid\_1\_AutoML\_20190911\_090503\_model\_23’

# Loading model to avoid training time again saved\_model = h2o.load\_model('C:\\model\_bankrupt\\GBM\_grid\_1\_AutoML\_20190911\_090503\_model\_23')

Once you have the model loaded, you can make predictions with it like this code below:

# Examlple on how to predict with loaded model saved\_model.predict(hf)

0\*2ob81CyjEdSerxaa?q=20

Let’s go over how to read that output above. Reach row is a prediction from the test set. The first column, ‘predict’, shows what the model predicted. In our case, 0 means not bankrupt. p0 is the probability the model thinks the prediction should be a 0, while p1 is the probability the model thinks the first row should be bankrupt. So, our GBM model thinks the first test company is 99% not going to be bankrupt. Cool, right?

Lastly, don’t forget to shut down your h2o session:

# Closing an h2o session after use h2o.shutdown()

0\*aYbkq9Ml3HZp7W6e?q=20

Conclusion

Awesome, we did a ton of things in this article! First, we discussed the importance of bankruptcy, and how we can apply this model to our own personal portfolio. Next, we loaded the data set of bankrupt Polish companies from Kaggle. Third, we applied logistic regression and xgboost just to get a baseline of how some models perform. Lastly, we automated an even better predictive model through channeling the streamlining powers of h2o!