Homework 2 Final Models (Loan Prediction)

Adam Kiehl
4/23/23

Setup

```
# import analysis packages
import keras
from keras.callbacks import EarlyStopping
from keras.layers import Dense, Dropout
from keras import models
from keras.regularizers import l2
from keras.utils import to_categorical
import keras.backend as back
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn
from sklearn.compose import make_column_selector
from sklearn.compose import make_column_transformer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
import tensorflow as tf
import tensorflow_addons as tfa
from tensorflow_addons.metrics import F1Score
```

```
/Users/akiehl/miniconda3/envs/dsci/lib/python3.8/site-packages/tensorflow_addons/utils/tfa_e

TensorFlow Addons (TFA) has ended development and introduction of new features.
TFA has entered a minimal maintenance and release mode until a planned end of life in May 20
Please modify downstream libraries to take dependencies from other repositories in our Tenso
```

```
For more information see: https://github.com/tensorflow/addons/issues/2807

  warnings.warn(
/Users/akiehl/miniconda3/envs/dsci/lib/python3.8/site-packages/tensorflow_addons/utils/ensure
 The versions of TensorFlow you are currently using is 2.9.2 and is not supported.
Some things might work, some things might not.
If you were to encounter a bug, do not file an issue.
If you want to make sure you're using a tested and supported configuration, either change the
You can find the compatibility matrix in TensorFlow Addon's readme:
https://github.com/tensorflow/addons
  warnings.warn(
```

Data Preparation

```
# read data from .csvs
trainDF = pd.read_csv('./loan_train.csv')
testDF = pd.read_csv('./loan_test.csv')
```

```
# separate response/prediction columns
trainResp = np.where(trainDF['MIS_Status'] == 'P I F', 1, 0)
trainDF.drop('MIS_Status', axis = 1, inplace = True)
testIDs = testDF['CustomerId']
testDF.drop('CustomerId', axis = 1, inplace = True)

# combine data sets for preprocessing
trainDF['source'] = 'train'
testDF['source'] = 'test'
fullDF = pd.concat([trainDF, testDF], axis = 0)

# factor categorical predictors
fullDF['NAICS'] = fullDF['NAICS'].apply(lambda x: str(x))
fullDF['NewExist'] = fullDF['NewExist'].apply(lambda x: str(x))
fullDF['UrbanRural'] = fullDF['UrbanRural'].apply(lambda x: str(x))
fullDF['RevLineCr'] = np.where(fullDF['RevLineCr'] == 'Y', 'Y', 'N')
fullDF['LowDoc'] = np.where(fullDF['LowDoc'] == 'Y', 'Y', 'N')
fullDF['New'] = fullDF['New'].apply(lambda x: str(x))
fullDF['RealEstate'] = fullDF['RealEstate'].apply(lambda x: str(x))
fullDF['Recession'] = fullDF['Recession'].apply(lambda x: str(x))

# selected predictors
```

```
predictors = ['NAICS', 'Term', 'NoEmp', 'CreateJob', 'RetainedJob', 'UrbanRural', 'RevLine
src = fullDF['source']
fullDF = fullDF[predictors]
```

```
# scale numeric predictors and encode categorical predictors
findNumPredictors = make_column_selector(dtype_exclude = object)
findCatPredictors = make_column_selector(dtype_include = object)
transform = make_column_transformer((MinMaxScaler(), findNumPredictors),
                                    (OneHotEncoder(), findCatPredictors))

# get new column names
colNames = transform.fit(fullDF).get_feature_names_out()

# transform data
modelDF = pd.DataFrame(transform.fit_transform(fullDF), columns = colNames)

# split data into training, validation, and test sets
modelTrain = modelDF.loc[np.where(src == 'train')]
modelTest = modelDF.iloc[np.where(src == 'test')]
X_train, X_valid, y_train, y_valid = train_test_split(modelTrain, trainResp, test_size = 0
```

Random Forest Classifier

```
# tuned hyperparameter
M = 38

# fit random forest model
model1 = RandomForestClassifier(max_features = M, n_estimators = 1000, random_state = 4192
model1.fit(X_train, y_train)

# predict on validation set
pred1 = model1.predict(X_valid)

# validation accuracy
display(f"Validation F1 score: {f1_score(pred1, y_valid).round(3)}")
```

```
'Validation F1 score: 0.865'
```

Gradient Boosted Classifier

```
# tuned hyperparameters
L = 0.1
D = 1

# fit gradient boosted model
model2 = GradientBoostingClassifier(learning_rate = L, max_depth = D, n_estimators = 1000,
model2.fit(X_train, y_train)

# predict on validation set
pred2 = model2.predict(X_valid)

# validation accuracy
display(f"Validation F1 score: {f1_score(pred2, y_valid).round(3)}")
```

```
'Validation F1 score: 0.871'
```

Neural Network

```
# set random seed
np.random.seed(462023)
tf.random.set_seed(482023)

# define F1 metric
f1_score_metric = F1Score(num_classes = 1, threshold = 0.5)

# penalty hyperparameter
PENALTY = 0.025

# define model architecture
model3 = models.Sequential([
    Dense(512, activation = 'relu', kernel_regularizer = l2(PENALTY), input_shape = (X_tra
    Dense(256, activation = 'relu', kernel_regularizer = l2(PENALTY)),
    Dense(128, activation = 'relu', kernel_regularizer = l2(PENALTY)),
    Dense(64, activation = 'relu', kernel_regularizer = l2(PENALTY)),
    Dense(32, activation = 'relu', kernel_regularizer = l2(PENALTY)),
    Dense(1, activation = 'sigmoid')
])
```

```
# compile model
model3.compile(optimizer = 'rmsprop',
               loss = 'binary_crossentropy',
               metrics = ['accuracy', f1_score_metric])

# model summary
model3.summary()

# number of epochs
EPOCHS = 50

# early stopping criteria
earlyStop = EarlyStopping(monitor = 'f1_score', mode = 'max', verbose = 1, patience = 3)

# train model
trained3 = model3.fit(X_train,
                      y_train,
                      epochs = EPOCHS,
                      batch_size = 64,
                      callbacks = earlyStop,
                      verbose = 0)

# predict on validation set
pred3 = model3.predict(X_valid)

# validation accuracy
display(f"Validation F1 score: {f1_score(pred3.round(), y_valid).round(3)}")
```

```
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_142 (Dense)           (None, 512)               23040

 dense_143 (Dense)           (None, 256)               131328

 dense_144 (Dense)           (None, 128)               32896

 dense_145 (Dense)           (None, 64)                8256

 dense_146 (Dense)           (None, 32)                2080
```

```
 dense_147 (Dense)           (None, 1)                 33

=================================================================
Total params: 197,633
Trainable params: 197,633
Non-trainable params: 0
_________________________________________________________________
Epoch 5: early stopping
7/7 [==============================] - 0s 10ms/step

'Validation F1 score: 0.809'
```

```
# set random seed
np.random.seed(462023)
tf.random.set_seed(482023)

# define F1 metric
f1_score_metric = F1Score(num_classes = 1, threshold = 0.5)

# penalty hyperparameter
RATE = 0.1

# define model architecture
model4 = models.Sequential([
    Dense(512, activation = 'relu', input_shape = (X_train.shape[1], )),
    Dropout(RATE),
    Dense(256, activation = 'relu'),
    Dropout(RATE),
    Dense(128, activation = 'relu'),
    Dropout(RATE),
    Dense(64, activation = 'relu'),
    Dropout(RATE),
    Dense(32, activation = 'relu'),
    Dropout(RATE),
    Dense(1, activation = 'sigmoid')
])

# compile model
model4.compile(optimizer = 'rmsprop',
               loss = 'binary_crossentropy',
               metrics = ['accuracy', f1_score_metric])
```

```
# model summary
model4.summary()

# number of epochs
EPOCHS = 50

# early stopping criteria
earlyStop = EarlyStopping(monitor = 'f1_score', mode = 'max', verbose = 1, patience = 3)

# train model
trained4 = model4.fit(X_train,
                      y_train,
                      epochs = EPOCHS,
                      batch_size = 64,
                      callbacks = earlyStop,
                      verbose = 0)

# predict on validation set
pred4 = model4.predict(X_valid)

# validation accuracy
display(f"Validation F1 score: {f1_score(pred4.round(), y_valid).round(3)}")
```

```
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_130 (Dense)           (None, 512)               23040

 dropout_70 (Dropout)        (None, 512)               0

 dense_131 (Dense)           (None, 256)               131328

 dropout_71 (Dropout)        (None, 256)               0

 dense_132 (Dense)           (None, 128)               32896

 dropout_72 (Dropout)        (None, 128)               0

 dense_133 (Dense)           (None, 64)                8256

 dropout_73 (Dropout)        (None, 64)                0
```

```
 dense_134 (Dense)           (None, 32)                2080

 dropout_74 (Dropout)        (None, 32)                0

 dense_135 (Dense)           (None, 1)                 33

=================================================================
Total params: 197,633
Trainable params: 197,633
Non-trainable params: 0
_________________________________________________________________
Epoch 34: early stopping
7/7 [==============================] - 0s 11ms/step


'Validation F1 score: 0.848'
```

Final Models

```
# tuned hyperparameters
L = 0.1
D = 1

# fit gradient boosted model
gbFit = GradientBoostingClassifier(learning_rate = L, max_depth = D, n_estimators = 1000,
gbFit.fit(modelTrain, trainResp)

# predict on test set
gbPred = gbFit.predict(modelTest)
```

```
# create submission data frame
submission = pd.DataFrame({'CustomerId': testIDs, 'Approve': gbPred})

# export submission
submission.to_csv('./submission1.csv', index = False)
```

```
# set random seed
np.random.seed(462023)
tf.random.set_seed(482023)
```

```
# define F1 metric
f1_score_metric = F1Score(num_classes = 1, threshold = 0.5)

# penalty hyperparameter
RATE = 0.1

# define model architecture
nnFit = models.Sequential([
    Dense(512, activation = 'relu', input_shape = (modelTrain.shape[1], )),
    Dropout(RATE),
    Dense(256, activation = 'relu'),
    Dropout(RATE),
    Dense(128, activation = 'relu'),
    Dropout(RATE),
    Dense(64, activation = 'relu'),
    Dropout(RATE),
    Dense(32, activation = 'relu'),
    Dropout(RATE),
    Dense(1, activation = 'sigmoid')
])

# compile model
nnFit.compile(optimizer = 'rmsprop',
              loss = 'binary_crossentropy',
              metrics = ['accuracy', f1_score_metric])

# model summary
nnFit.summary()

# number of epochs
EPOCHS = 50

# early stopping criteria
earlyStop = EarlyStopping(monitor = 'f1_score', mode = 'max', verbose = 1, patience = 3)

# train model
nnFit.fit(modelTrain,
          trainResp,
          epochs = EPOCHS,
          batch_size = 64,
          callbacks = earlyStop,
```

```
            verbose = 0)

  # predict on test set
  nnPred = nnFit.predict(modelTest).round().astype(int)
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_76 (Dense)            (None, 512)               23040

 dropout_47 (Dropout)        (None, 512)               0

 dense_77 (Dense)            (None, 256)               131328

 dropout_48 (Dropout)        (None, 256)               0

 dense_78 (Dense)            (None, 128)               32896

 dropout_49 (Dropout)        (None, 128)               0

 dense_79 (Dense)            (None, 64)                8256

 dense_80 (Dense)            (None, 32)                2080

 dense_81 (Dense)            (None, 1)                 33

=================================================================
Total params: 197,633
Trainable params: 197,633
Non-trainable params: 0
_________________________________________________________________
Epoch 21: early stopping
32/32 [==============================] - 0s 3ms/step
```

```
  # create submission data frame
  submission = pd.DataFrame({'CustomerId': testIDs, 'Approve': nnPred.reshape(len(nnPred), )

  # export submission
  submission.to_csv('./submission2.csv', index = False)
```