Bryce Smith
STAA 578
Homework 2 Write-up

The goal of homework 2 was to predict whether or not the bank should loan money to an applicant. The training dataset initially consisted of 31 columns and 1102 rows. When exploring the explanatory variables of this dataset, it was clear that some of the columns needed to change datatype while some of the columns could be dropped. I chose to drop the following columns due to not being needed for prediction: *LoanNr_ChkDgt*, *Name*, *daysterm*, *xx*. My decisions on what datatype a column should be and what columns weren't needed were derived from exploring the meta data information provided on Kaggle. After cleaning up the explanatory variables of the dataset and separating the predictor variable, the training dataset consisted of 26 columns.

To preprocess the data, I split the explanatory variables into categorical and numeric lists. To pre-process the categorical data I called an instance of *OneHotEncoder* with the *handle_unknown* parameter set to ignore so that when performing these steps on the test data, any unknown categories will be ignored and not cause error in the model. The resulting columns are a set of binary columns for each categorical explanatory variable. Lastly, I standardized the numeric variables by using *StandardScaler* which removes the mean and scaling to unit variance.

I took a combination of validation and dropout approaches to train the neural network model. To begin I instantiated a sequential model and then created 3 dense layers and 3 dropout layors, ending with a dense output later with a single neuron with sigmoid activation for binary classification. The first three dense layers used a *ReLu* activation functions to allow the model to learn complex patterns and approximations. Lastly, each dropout layer used a rate of 0.5. To compile the model I used the Adam optimizer after researching strong optimizers for training binary classification neural networks. I set the loss function to *binary_crossentropy* and the evaluation metric to *accuracy*. Lastly, as I trained the model using the *fit* method, I specified a validation split of 0.2, meaning 20% of the training data was held out for validation to assess the model's performance during training.

The neural network model described above performed well, resulting in a 0.70924 score on the test dataset, evaluated in Kaggle. Though I was happy with this performance, I also decided to train an ensemble learning method that I know has worked well on classification models in the past. I used a Gradient Boosting Classifier model, which iteratively adds decisions trees in an ensemble method to minimize the loss function. To train this model I used two different sets of parameters. The first set was a default set of parameters with *n_estimators* set to 100, *max_depth* set to 3, and *learning_rate* set to 0.1. In an effort to improve this model, I used a grid search to optimize these parameters. The result was *n_estimators* set to 50, *max_depth* set to 3, and *learning_rate* set to 0.15. The default parameters produced a score of 0.79787, while the optimized parameters scored 0.78773. I proceeded with default parameters.

In [ ]:

```python
#Load the necessary packages
import numpy as np
import pandas as pd
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from tensorflow.keras import layers, models
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
```

In [ ]:

```python
#Read in training data
df_train = pd.read_csv("loan_train.csv")
dF_metaData = pd.read_csv("Meta_Data.csv")
```

In [ ]:

```python
#Read in test data
df_test = pd.read_csv("loan_test.csv")
```

In [ ]:

```python
#Change data types of explanatory variables
df_train['ApprovalDate'] = pd.to_datetime(df_train['ApprovalDate'])
df_train['DisbursementDate'] = pd.to_datetime(df_train['DisbursementDate'])
df_train['City'] = df_train['City'].astype('category')
df_train['State'] = df_train['State'].astype('category')
df_train['Bank'] = df_train['Bank'].astype('category')
df_train['BankState'] = df_train['BankState'].astype('category')
df_train['NAICS'] = df_train['NAICS'].astype('category')
df_train['Zip'] = df_train['Zip'].astype('category')
df_train['NewExist'] = df_train['NewExist'].astype('category')
df_train['FranchiseCode'] = df_train['FranchiseCode'].astype('category')
df_train['UrbanRural'] = df_train['UrbanRural'].astype('category')
df_train['RevLineCr'] = df_train['RevLineCr'].astype('category')
df_train['LowDoc'] = df_train['LowDoc'].astype('category')
df_train['New'] = df_train['New'].astype('category')
df_train['RealEstate'] = df_train['RealEstate'].astype('category')
df_train['Recession'] = df_train['Recession'].astype('category')
df_train['ApprovalFY'] = df_train['ApprovalFY'].astype('category')

#drop unused columns
df_train.drop('LoanNr_ChkDgt', axis=1, inplace=True)
df_train.drop('Name', axis=1, inplace=True)
df_train.drop('daysterm', axis=1, inplace=True)
df_train.drop('xx', axis=1, inplace=True)
```

In [ ]:

```python
#Prepare to pre-Process training Data
df_train['MIS_Status_Resp'] = df_train['MIS_Status'].map({'CHGOFF':1, 'P I F':0}
train_x = df_train.drop(['MIS_Status', 'MIS_Status_Resp'], axis = 1)
train_y = df_train['MIS_Status_Resp']

numeric = ['Term', 'NoEmp','CreateJob', 'RetainedJob','DisbursementGross', 'Bala
           'SBA_Appv', 'Portion']

categorical = ['Zip', 'NAICS', 'ApprovalFY', 'City', 'State', 'Bank', 'BankState
               'FranchiseCode','UrbanRural','New', 'RealEstate','RevLineCr', 'Lo

date = ['ApprovalDate', 'DisbursementDate']
```

In [ ]:

```
train_x.shape
```

In [ ]:

```
#Pre-process Categorical data
encoder = OneHotEncoder(handle_unknown='ignore')
cat_data = encoder.fit_transform(df_train[categorical])
transformed_columns = encoder.get_feature_names_out(categorical)
cat_data = pd.DataFrame(cat_data.toarray(), columns=transformed_columns)
```

In [ ]:

```
#Pre-process numeric data
scaler = StandardScaler()
numeric_data = pd.DataFrame(scaler.fit_transform(df_train[numeric]), columns=num

date = pd.DataFrame(date)
```

In [ ]:

```
#Combine pre-processed data into a dataframe to train the model
train_x = pd.concat([numeric_data, cat_data], axis=1)
```

In [ ]:

```
#Define the neural network
model = models.Sequential()
model.add(layers.Dense(120, activation='relu', input_shape=(train_x.shape[1],)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(60, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='Adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [ ]:

```
#Train the model and store history
history = model.fit(train_x, train_y, epochs = 10, batch_size = 32, validation_s
```

In [ ]:

```
#Evaluate train model on training dataset
test_loss, test_acc = model.evaluate(train_x, train_y, verbose=2)
```

In [ ]:

```
#Change data types of explanatory variables for test data
df_test['ApprovalDate'] = pd.to_datetime(df_test['ApprovalDate'])
df_test['DisbursementDate'] = pd.to_datetime(df_test['DisbursementDate'])
df_test['City'] = df_test['City'].astype('category')
df_test['State'] = df_test['State'].astype('category')
df_test['Bank'] = df_test['Bank'].astype('category')
df_test['BankState'] = df_test['BankState'].astype('category')
df_test['NAICS'] = df_test['NAICS'].astype('category')
df_test['Zip'] = df_test['Zip'].astype('category')
df_test['NewExist'] = df_test['NewExist'].astype('category')
df_test['FranchiseCode'] = df_test['FranchiseCode'].astype('category')
df_test['UrbanRural'] = df_test['UrbanRural'].astype('category')
df_test['RevLineCr'] = df_test['RevLineCr'].astype('category')
df_test['LowDoc'] = df_test['LowDoc'].astype('category')
df_test['New'] = df_test['New'].astype('category')
df_test['RealEstate'] = df_test['RealEstate'].astype('category')
df_test['Recession'] = df_test['Recession'].astype('category')
df_test['ApprovalFY'] = df_test['ApprovalFY'].astype('category')
```

```
#Drop unused colummns
df_test.drop('LoanNr_ChkDgt', axis=1, inplace=True)
df_test.drop('Name', axis=1, inplace=True)
df_test.drop('daysterm', axis=1, inplace=True)
df_test.drop('xx', axis=1, inplace=True)
```

In [ ]:

```
#Preprocess Test Data following same steps taken on training data
cat_data = encoder.transform(df_test[categorical])
transformed_columns = encoder.get_feature_names_out(categorical)
cat_data = pd.DataFrame(cat_data.toarray(), columns=transformed_columns)

numeric_data = pd.DataFrame(scaler.transform(df_test[numeric]), columns=numeric)

test_x = pd.concat([numeric_data, cat_data], axis=1)
```

In [ ]:

```
#Use the trained model on the test data explanatory variables
predictions = model.predict(test_x)
```

In [ ]:

```
#Convert predictions to binary column then format for export
df_test['Approve'] = (1 - np.round(predictions)).astype(int)
my_predictions = df_test[['CustomerId', 'Approve']]
```

In [ ]:

```
my_predictions.head()
```

In [ ]:

```
#Export predictions
my_predictions.to_csv("HW2_Preds_NN_2.csv", index=False)
```

In [ ]:

```
# grid = {
#     'n_estimators': [10, 25, 50, 100, 125, 150],
#     'max_depth': [2, 3, 4, 5, 6, 8],
#     'learning_rate': [0.025, 0.05, 0.75, 0.1, 0.125, 0.15]
# }

# search = GridSearchCV(estimator=GradientBoostingClassifier(random_state=99),
#     param_grid=grid, cv=2, n_jobs=-1, scoring='accuracy'
# )

# search.fit(train_x, train_y)
```

In [ ]:

```
# print('Best Parameters:', gridsearch.best_params_)
# print('Best Score:', gridsearch.best_score_)
```

In [ ]:

```
#Create Gradient Boosting Classifier Model
#Highest Accuracy so far
grad_boost = GradientBoostingClassifier(n_estimators=100, max_depth=3, learning_
#Fit Gradient Boosting Classifier Model
grad_boost.fit(train_x,train_y)
```

In [ ]:

```
#Develop predictions using test data and gradient boosting classifier
y_Pred_grad_boost = grad_boost.predict(test_x)
```

In [ ]:

```
#Convert predictions to binary column then format for export
df_test['Approve'] = (1 - np.round(y_Pred_grad_boost.astype(int)))
my_predictions = df_test[['CustomerId', 'Approve']]
#my_predictions.head()
```

In [ ]:

```
#Export predictions
my_predictions.to_csv("HW2_Preds_GradBoostNew_Rev.csv", index=False)
```