Import Needed Modules

In [1]:
```
#general libraries needed
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#TensorFlow requirements
import tensorflow as tf
from tensorflow import keras

#scikit learn imports
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder,OrdinalEncoder, StandardScaler, MinMaxS
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predic
```

In [2]:
```
#display version of TensorFlow - must be newer than 2.4.0
tf.__version__
```

Out[2]:
```
'2.8.0'
```

Function Definitions

In [3]:
```
#function to verify the existence of a file in the current working directory and downlo
import os,urllib, urllib.request, sys, tarfile
def downloadDataResource(file,sourcePath,compressed=None):
    if not os.path.isfile(file):
        try:
            urllib.request.urlretrieve(sourcePath+(compressed if compressed else file),
            print("Downloaded", (compressed if compressed else file) )
            if compressed:
                ucomp = tarfile.open(compressed)
                ucomp.extractall()
                ucomp.close()
                print("File uncompressed.")
        except:
            print("ERROR: File", (compressed if compressed else file), "not found. Data
    else:
        print("Data resource", file, "already downloaded.")
```

In [4]:
```
#function that shows a learning curve for any model that has predict or fit methods
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator,X,y,ylim=None,cv=None,n_jobs=None,train_sizes=np.lins

    _, axes = plt.subplots(1, 1, figsize=(10, 5))
    axes.set_title('Learning Curve')
    if ylim is not None:
        axes.set_ylim(*ylim)
```

```
    axes.set_xlabel("Training examples")
    axes.set_ylabel(scoring)

    train_sizes, train_scores, test_scores= learning_curve(estimator,X,y,cv=cv,n_jobs=n
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    # Plot learning curve
    axes.grid()
    axes.fill_between(train_sizes,train_scores_mean - train_scores_std,train_scores_mea
    axes.fill_between(train_sizes,test_scores_mean - test_scores_std,test_scores_mean +
    axes.plot(train_sizes, train_scores_mean, "o-", color="r", label="Training score")
    axes.plot(train_sizes, test_scores_mean, "o-", color="g", label="Cross-validation s
    axes.legend(loc="best")
    plt.show()

    return

#code to prevent warnings that can occur as a result of this function
from warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning
simplefilter("ignore", category=ConvergenceWarning)
```

In [5]:

```
#function provided that plots the learning curve for neural networks
def nn_plot_learning_curve( history ):
    pd.DataFrame(history.history).plot(figsize=(8, 5))
    plt.grid(True)
    ymin, ymax = [], []
    for x in history.history.keys():
        ymax.append( max(history.history[x]))
        ymin.append( min(history.history[x]))
    plt.gca().set_ylim(min(ymin)*.95, max(ymax)*1.05)
    plt.xlabel("EPOCHS")
    plt.show()
```

Source Data

In [6]:

```
#download data files if not currently downloaded into the current working directory
path = "https://raw.githubusercontent.com/SueMcMetzger/MachineLearning/main/chpt8/"

filename = ["In-VehicleCoupon.csv"]

for f in filename:
    downloadDataResource(f,path)
```

```
Data resource In-VehicleCoupon.csv already downloaded.
```

In [7]:

```
#read file into dataframe
data = pd.read_csv(filename[0])
```

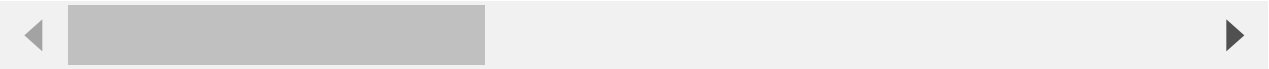
Prepare Data

In [8]:
```
data.head()
```

Out[8]:

	destination	passenger	weather	temperature	time	coupon	expiration	gender	age	mari
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	Female	21	U
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	Female	21	U
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	Female	21	U
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	2h	Female	21	U
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	Female	21	U

5 rows × 25 columns



In [9]:
```
# replace gender with binary values
# 0 = female
# 1 = male
data['Male'] = data.apply( lambda x: 0 if x.gender == "Female" else 1, axis=1)

#drop gender
data.drop(columns = ['gender'], inplace=True)
```

In [10]:
```
print(data.isnull().sum())
```

```
destination               0
passenger                 0
weather                   0
temperature               0
time                      0
coupon                    0
expiration                0
age                       0
maritalStatus             0
has_children              0
education                 0
occupation                0
income                    0
Bar                     107
CoffeeHouse             217
CarryAway               151
RestaurantLessThan20    130
Restaurant20To50        189
toCoupon_5min             0
toCoupon_15min            0
```

```
toCoupon_25min            0
direction_same            0
direction_opp             0
AcceptCoupon              0
Male                      0
dtype: int64
```

MISSING DATA - imputate NaN instances w/ most frequent

- Bar
- CoffeeHouse
- CarryAway
- RestaurantLessThan20
- Restaurant20To50

In [11]:

```
# filling NaN values with most common class
data = data.apply(lambda x: x.fillna(x.value_counts().index[0]))

print(data.isnull().sum())
```

```
destination             0
passenger               0
weather                 0
temperature             0
time                    0
coupon                  0
expiration              0
age                     0
maritalStatus           0
has_children            0
education               0
occupation              0
income                  0
Bar                     0
CoffeeHouse             0
CarryAway               0
RestaurantLessThan20    0
Restaurant20To50        0
toCoupon_5min           0
toCoupon_15min          0
toCoupon_25min          0
direction_same          0
direction_opp           0
AcceptCoupon            0
Male                    0
dtype: int64
```

Split Data

In [12]:

```
#split the data using standard train_test_split ()
#drop direction_same since already measured by direction_opp
#drop toCoupon_5min since no variance in feature

X_train, X_test, y_train, y_test = train_test_split(
    data.drop(columns=['direction_same', 'toCoupon_5min','AcceptCoupon']),
```

```
    data['AcceptCoupon'],
    random_state=42
)
```

Transform Data

Ordinal Encoding - Category Orders

In [13]:
```
ordinal_attribs = ['time', 'age', 'education', 'income',
                   'Bar', 'CoffeeHouse', 'CarryAway',
                   'RestaurantLessThan20', 'Restaurants20To50']
```

In [14]:
```
#unique values - time
data['time'].unique()
```

Out[14]:
```
array(['2PM', '10AM', '6PM', '7AM', '10PM'], dtype=object)
```

In [15]:
```
# set order - time
hour = ['7AM', '10AM', '2PM', '6PM', '10PM']
```

In [16]:
```
# unique values - age
data['age'].unique()
```

Out[16]:
```
array(['21', '46', '26', '31', '41', '50plus', '36', 'below21'],
      dtype=object)
```

In [17]:
```
# set order - age
ages = ['below21','21','26','31','36','41','46', '50plus']
```

In [18]:
```
# unique values - education
data['education'].unique()
```

Out[18]:
```
array(['Some college - no degree', 'Bachelors degree',
       'Associates degree', 'High School Graduate',
       'Graduate degree (Masters or Doctorate)', 'Some High School'],
      dtype=object)
```

In [19]:
```
# set order - education
edu = ['Some High School', 'High School Graduate',
       'Some college - no degree', 'Associates degree',
       'Bachelors degree','Graduate degree (Masters or Doctorate)']
```

In [20]:
```
# unique values - income
data['income'].unique()
```

Out[20]:
```
array(['$37500 - $49999', '$62500 - $74999', '$12500 - $24999',
       '$75000 - $87499', '$50000 - $62499', '$25000 - $37499',
```

```
       '$100000 or More', '$87500 - $99999', 'Less than $12500'],
      dtype=object)
```

In [21]:
```
# set order - income
pay = ['Less than $12500','$12500 - $24999','$25000 - $37499',
       '$37500 - $49999','$50000 - $62499','$62500 - $74999',
       '$75000 - $87499','$87500 - $99999','$100000 or More',  ]
```

In [22]:
```
# unique values - Bar
data['Bar'].unique()
```

Out[22]:
```
array(['never', 'less1', '1~3', 'gt8', '4~8'], dtype=object)
```

In [23]:
```
# set order - Bar
bars = ['never', 'less1', '1~3','4~8', 'gt8' ]
```

In [24]:
```
# unique values - CoffeeHouse
data['CoffeeHouse'].unique()
```

Out[24]:
```
array(['never', 'less1', '4~8', '1~3', 'gt8'], dtype=object)
```

In [25]:
```
# set order - CoffeeHouse
coffee = ['never', 'less1', '1~3','4~8', 'gt8' ]
```

In [26]:
```
# unique values - CarryAway
data['CarryAway'].unique()
```

Out[26]:
```
array(['1~3', '4~8', 'gt8', 'less1', 'never'], dtype=object)
```

In [27]:
```
# set order - CarryAway
takeout = ['never', 'less1', '1~3','4~8', 'gt8' ]
```

In [28]:
```
# unique values - RestaurantLessThan20
data['RestaurantLessThan20'].unique()
```

Out[28]:
```
array(['4~8', '1~3', 'less1', 'gt8', 'never'], dtype=object)
```

In [29]:
```
# set order - RestaurantLessThan20
LessThan20 = ['never', 'less1', '1~3','4~8', 'gt8' ]
```

In [30]:
```
# unique values - Restaurant20To50
data['Restaurant20To50'].unique()
```

Out[30]:
```
array(['1~3', 'less1', 'never', 'gt8', '4~8'], dtype=object)
```

In [31]:

```
# set order - Restaurant20To50
TwentyTo50 = ['never', 'less1', '1~3','4~8', 'gt8' ]
```

Run Transformation Pipeline

In [32]:

```
#set categorical attributes

#ALL ATTRIBUTES
cat_attribs = ['destination', 'passenger', 'weather', 'time', 'coupon', 'expiration','a

#ORDINAL ATTRIBUTES
ordinal_attribs = ['time', 'age', 'education', 'income', 'Bar', 'CoffeeHouse', 'CarryAw
                   'RestaurantLessThan20', 'Restaurant20To50']

#ONE HOT ATTRIBUTES
oneHot_attribs = ['destination', 'passenger', 'weather', 'coupon', 'expiration', 'marit


#set the numerical attributes
num_attribs = list( X_train.drop(cat_attribs, axis = 1) )


#define pipeline for numeric attributes
#each numeric attribute will be imputated using the Median strategy
#each numeric attribute will be scaled
num_pipeline = Pipeline( [
    ('imputer', SimpleImputer(missing_values=np.nan, strategy="median")),
    ('std_scaler', MinMaxScaler()),
])

#define the pipeline process for the data set
full_pipeline = ColumnTransformer( [
    ('num', num_pipeline, num_attribs),
    ('hot', OneHotEncoder(sparse=False), oneHot_attribs),
    ('time_ord', OrdinalEncoder(categories= [hour]), ['time']),
    ('age_ord', OrdinalEncoder(categories= [ages]), ['age']),
    ('edu_ord', OrdinalEncoder(categories= [edu]), ['education']),
    ('inc_ord', OrdinalEncoder(categories= [pay]), ['income']),
    ('bar_ord', OrdinalEncoder(categories= [bars]), ['Bar']),
    ('coffee_ord', OrdinalEncoder(categories= [coffee]), ['CoffeeHouse']),
    ('carry_ord', OrdinalEncoder(categories= [takeout]), ['CarryAway']),
    ('Less20_ord', OrdinalEncoder(categories= [LessThan20]), ['RestaurantLessThan20']),
    ('TwentyTo50_ord', OrdinalEncoder(categories= [TwentyTo50]), ['Restaurant20To50'])
])
```

Create X Data Sets

In [33]:

```
X_train = full_pipeline.fit_transform( X_train )
X_test = full_pipeline.transform( X_test )

X_train.shape, X_test.shape
```

Out[33]: ((9513, 62), (3171, 62))

Best Non-Neural Model:

In [53]:
```
#create a parameter grid that determines the variable hyperparameters
param_grid = [
    {'solver': ['saga', 'newton-cg', 'lbfgs', 'sag'],
     'multi_class': ['ovr', 'multinomial'],
     'C': [3, 1.5, 2]
    }
]

# train across 10 folds
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=10,
                           scoring='accuracy',
                           return_train_score=True)

#fit data to the model
grid_search.fit(X_train, y_train)
```

Out[53]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid=[{'C': [3, 1.5, 2],
                          'multi_class': ['ovr', 'multinomial'],
                          'solver': ['saga', 'newton-cg', 'lbfgs', 'sag']}],
             return_train_score=True, scoring='accuracy')
```

In [54]:
```
#display the results of the GridSearchCV
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print("Accuracy: {:,.2f}% - {}".format( mean_score*100, params) )
```

```
Accuracy: 67.78% - {'C': 3, 'multi_class': 'ovr', 'solver': 'saga'}
Accuracy: 67.78% - {'C': 3, 'multi_class': 'ovr', 'solver': 'newton-cg'}
Accuracy: 67.76% - {'C': 3, 'multi_class': 'ovr', 'solver': 'lbfgs'}
Accuracy: 67.78% - {'C': 3, 'multi_class': 'ovr', 'solver': 'sag'}
Accuracy: 67.75% - {'C': 3, 'multi_class': 'multinomial', 'solver': 'saga'}
Accuracy: 67.75% - {'C': 3, 'multi_class': 'multinomial', 'solver': 'newton-cg'}
Accuracy: 67.74% - {'C': 3, 'multi_class': 'multinomial', 'solver': 'lbfgs'}
Accuracy: 67.75% - {'C': 3, 'multi_class': 'multinomial', 'solver': 'sag'}
Accuracy: 67.78% - {'C': 1.5, 'multi_class': 'ovr', 'solver': 'saga'}
Accuracy: 67.78% - {'C': 1.5, 'multi_class': 'ovr', 'solver': 'newton-cg'}
Accuracy: 67.73% - {'C': 1.5, 'multi_class': 'ovr', 'solver': 'lbfgs'}
Accuracy: 67.78% - {'C': 1.5, 'multi_class': 'ovr', 'solver': 'sag'}
Accuracy: 67.78% - {'C': 1.5, 'multi_class': 'multinomial', 'solver': 'saga'}
Accuracy: 67.78% - {'C': 1.5, 'multi_class': 'multinomial', 'solver': 'newton-cg'}
Accuracy: 67.75% - {'C': 1.5, 'multi_class': 'multinomial', 'solver': 'lbfgs'}
Accuracy: 67.78% - {'C': 1.5, 'multi_class': 'multinomial', 'solver': 'sag'}
Accuracy: 67.79% - {'C': 2, 'multi_class': 'ovr', 'solver': 'saga'}
Accuracy: 67.79% - {'C': 2, 'multi_class': 'ovr', 'solver': 'newton-cg'}
Accuracy: 67.73% - {'C': 2, 'multi_class': 'ovr', 'solver': 'lbfgs'}
Accuracy: 67.79% - {'C': 2, 'multi_class': 'ovr', 'solver': 'sag'}
Accuracy: 67.76% - {'C': 2, 'multi_class': 'multinomial', 'solver': 'saga'}
Accuracy: 67.76% - {'C': 2, 'multi_class': 'multinomial', 'solver': 'newton-cg'}
Accuracy: 67.72% - {'C': 2, 'multi_class': 'multinomial', 'solver': 'lbfgs'}
Accuracy: 67.76% - {'C': 2, 'multi_class': 'multinomial', 'solver': 'sag'}
```

In [55]:
```
#display the best solution
grid_search.best_params_
```

Out[55]:
```
{'C': 2, 'multi_class': 'ovr', 'solver': 'saga'}
```

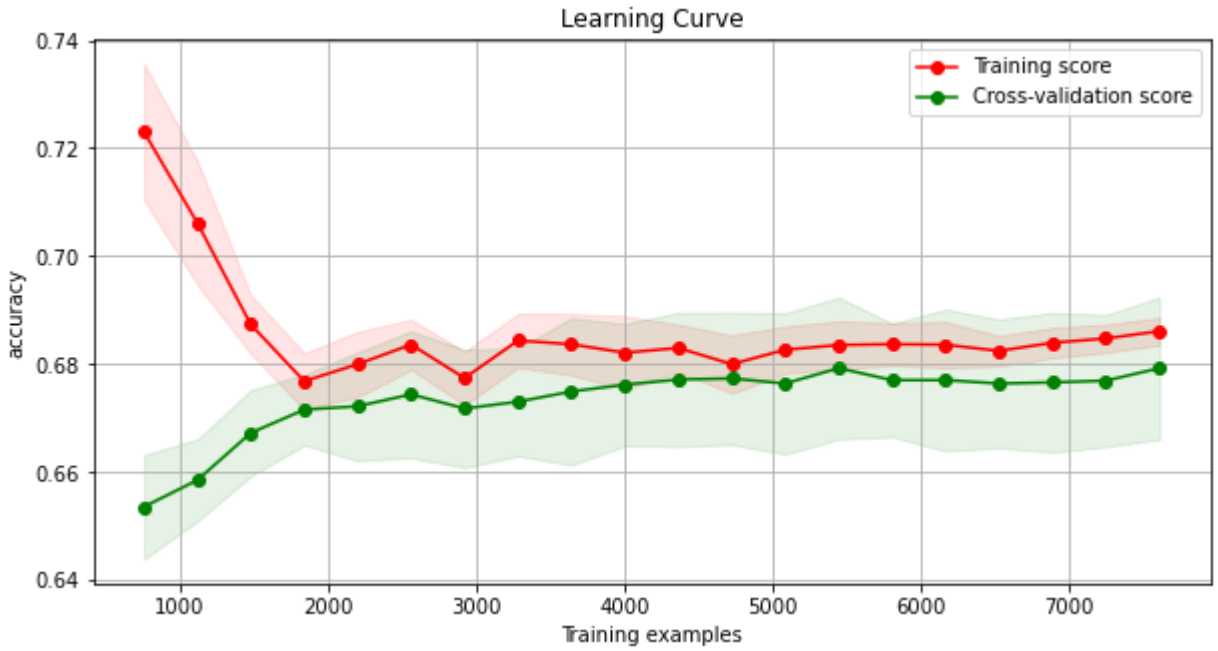
In [56]:
```
#use the above parameters to create the model
model = LogisticRegression(**grid_search.best_params_)

#fit model to the training data set
model.fit(X_train, y_train)

#calculate the accuracy (i.e. predicted vs. actual for the training data set)
acc = model.score(X_train, y_train)
print('Accuracy: {:.2f}%'.format(acc*100))
```

```
Accuracy: 68.34%
```

In [57]:
```
scores = cross_val_score(model, X_train, y_train, cv=10)
scores.mean()
print('Accuracy: {:.2f}%'.format(scores.mean()*100))
```

```
Accuracy: 67.79%
```

In [58]:
```
plot_learning_curve(model, X_train, y_train, scoring='accuracy')
```



In [59]:
```
#calculate the accuracy (i.e. predicted vs. actual for the test data set)
acc = model.score(X_test, y_test)
print('Accuracy: {:.2f}%'.format(acc*100))
```

```
Accuracy: 68.46%
```

Build Neural Network, Compile, and Train

In [80]:
```
#starting by setting random seeds and restarting keras blackend session
np.random.seed(42)
```

```
tf.random.set_seed(42)

#resets the Keras global state - helps avoid clutter from old models and layers, especi
keras.backend.clear_session()
```

In [81]:

```
#the same steps above can be built and saved in a single command with the same results
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=X_train.shape [1:]),
    keras.layers.Dense(60, activation="relu"),
    keras.layers.Dropout(.2),
    keras.layers.Dense(60, activation="relu"),
    keras.layers.Dropout(.2),
    keras.layers.Dense(50, activation="relu"),
    keras.layers.Dropout(.2),
    keras.layers.Dense(25, activation="relu"),
    keras.layers.Dropout(.2),
    keras.layers.Dense(1, activation="sigmoid")
])
```

In [82]:

```
#After model is created, it needs to be compiled - this requires setting
#the loss function for binary classification to binary_crossentropy
#but could be sparse_categorical_crossentropy if last layer is 2

model.compile(loss='binary_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=.0003),
              metrics=['accuracy']
             )
```

In [83]:

```
#fit the model and capture the details of the fit to a variable called history
#note that validation data will be created dynamically as 20% of the training data set

early_stopping = keras.callbacks.EarlyStopping( monitor='val_loss', mode='min', patienc

history = model.fit(X_train,
                    y_train,
                    epochs=500,
                    validation_split=.2,
                    callbacks=[early_stopping]
                   )
```

```
Epoch 1/500
238/238 [==============================] - 3s 8ms/step - loss: 0.6971 - accuracy: 0.5534
- val_loss: 0.6737 - val_accuracy: 0.6054
Epoch 2/500
238/238 [==============================] - 1s 5ms/step - loss: 0.6795 - accuracy: 0.5745
- val_loss: 0.6603 - val_accuracy: 0.6253
Epoch 3/500
238/238 [==============================] - 1s 5ms/step - loss: 0.6613 - accuracy: 0.6055
- val_loss: 0.6417 - val_accuracy: 0.6521
Epoch 4/500
238/238 [==============================] - 1s 4ms/step - loss: 0.6484 - accuracy: 0.6353
- val_loss: 0.6309 - val_accuracy: 0.6600
Epoch 5/500
238/238 [==============================] - 1s 5ms/step - loss: 0.6364 - accuracy: 0.6523
- val_loss: 0.6160 - val_accuracy: 0.6726
Epoch 6/500
```

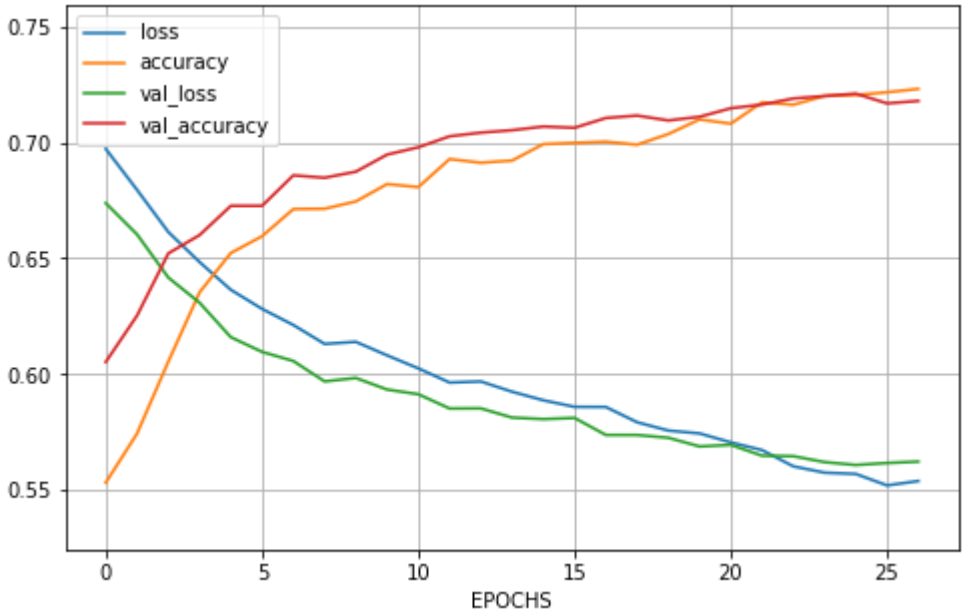
```
238/238 [==============================] - 1s 5ms/step - loss: 0.6282 - accuracy: 0.6595
- val_loss: 0.6097 - val_accuracy: 0.6726
Epoch 7/500
238/238 [==============================] - 1s 5ms/step - loss: 0.6213 - accuracy: 0.6712
- val_loss: 0.6057 - val_accuracy: 0.6858
Epoch 8/500
238/238 [==============================] - 1s 5ms/step - loss: 0.6131 - accuracy: 0.6714
- val_loss: 0.5970 - val_accuracy: 0.6847
Epoch 9/500
238/238 [==============================] - 1s 5ms/step - loss: 0.6140 - accuracy: 0.6745
- val_loss: 0.5984 - val_accuracy: 0.6873
Epoch 10/500
238/238 [==============================] - 1s 5ms/step - loss: 0.6082 - accuracy: 0.6820
- val_loss: 0.5934 - val_accuracy: 0.6947
Epoch 11/500
238/238 [==============================] - 1s 5ms/step - loss: 0.6026 - accuracy: 0.6807
- val_loss: 0.5914 - val_accuracy: 0.6978
Epoch 12/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5965 - accuracy: 0.6928
- val_loss: 0.5853 - val_accuracy: 0.7026
Epoch 13/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5969 - accuracy: 0.6912
- val_loss: 0.5853 - val_accuracy: 0.7042
Epoch 14/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5925 - accuracy: 0.6921
- val_loss: 0.5814 - val_accuracy: 0.7052
Epoch 15/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5888 - accuracy: 0.6992
- val_loss: 0.5807 - val_accuracy: 0.7068
Epoch 16/500
238/238 [==============================] - 1s 6ms/step - loss: 0.5860 - accuracy: 0.6997
- val_loss: 0.5812 - val_accuracy: 0.7063
Epoch 17/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5859 - accuracy: 0.7003
- val_loss: 0.5739 - val_accuracy: 0.7105
Epoch 18/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5794 - accuracy: 0.6989
- val_loss: 0.5738 - val_accuracy: 0.7115
Epoch 19/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5758 - accuracy: 0.7035
- val_loss: 0.5727 - val_accuracy: 0.7094
Epoch 20/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5745 - accuracy: 0.7099
- val_loss: 0.5689 - val_accuracy: 0.7110
Epoch 21/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5707 - accuracy: 0.7080
- val_loss: 0.5696 - val_accuracy: 0.7147
Epoch 22/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5673 - accuracy: 0.7172
- val_loss: 0.5649 - val_accuracy: 0.7162
Epoch 23/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5604 - accuracy: 0.7162
- val_loss: 0.5647 - val_accuracy: 0.7189
Epoch 24/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5576 - accuracy: 0.7200
- val_loss: 0.5622 - val_accuracy: 0.7199
Epoch 25/500
238/238 [==============================] - 1s 5ms/step - loss: 0.5571 - accuracy: 0.7204
- val_loss: 0.5609 - val_accuracy: 0.7210
Epoch 26/500
```

```
238/238 [==============================] - 1s 5ms/step - loss: 0.5521 - accuracy: 0.7216
- val_loss: 0.5618 - val_accuracy: 0.7168
Epoch 27/500
238/238 [==============================] - 2s 7ms/step - loss: 0.5540 - accuracy: 0.7230
- val_loss: 0.5624 - val_accuracy: 0.7178
```

Evaluate Performance

In [84]:

```
#plot the loss learning curve
nn_plot_learning_curve(history)
```



Curves haven't finished plateauing, possibly a little more learning to happen. Good convergence of curves

In [85]:

```
#evaluate Test set
accuracy_test = model.evaluate(X_test, y_test)

print('Loss: {:.2f}  ---  Accuracy: {:.2f}%'.format( accuracy_test[0], accuracy_test[1]
```

```
100/100 [==============================] - 0s 3ms/step - loss: 0.5502 - accuracy: 0.7304
Loss: 0.55  ---  Accuracy: 73.04%
```

In [ ]: