Problem 1A

Prepare the data

In [1]:

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split,cross_val_score,GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
import matplotlib.pyplot as plt
from sklearn import metrics
import matplotlib.pyplot as plt
import scikitplot as skplt
from matplotlib import style
style.use("fivethirtyeight")
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.metrics import r2_score
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
data=pd.read_excel('HW3.xlsx')
X=data.iloc[:,0:-2]
y=data.iloc[:,-1]
```

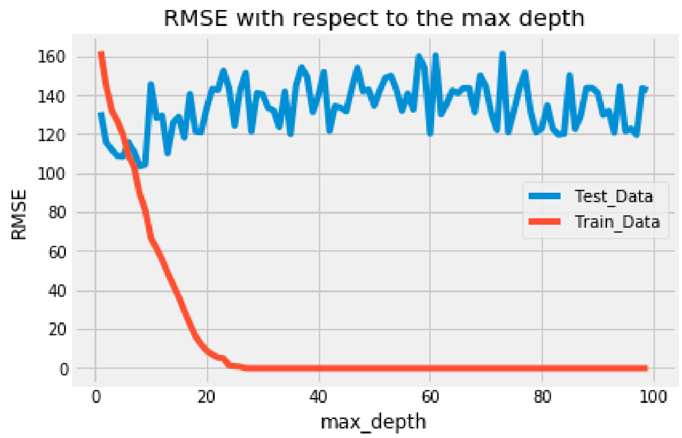
In [3]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=0)
```

Regression Tree

In [4]:

```
from sklearn.tree import DecisionTreeRegressor

fig = plt.figure()
ax0 = fig.add_subplot(111)
RMSE_train = []
RMSE_test = []
for i in range(1,100):
    #Paramterize the model and let i be the number of minimum instances per leaf node
    regression_model = DecisionTreeRegressor(criterion="mse",max_depth=i)
    #Train the model
    regression_model.fit(X_train,y_train)
    #Predict query instances
    predicted_train = regression_model.predict(X_train)
    predicted_test = regression_model.predict(X_test)
    #Calculate and append the RMSEs
    RMSE_train.append(np.sqrt(np.sum(((y_train-predicted_train)**2)/len(y_train))))
    RMSE_test.append(np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))

ax0.plot(range(1,100),RMSE_test,label='Test_Data')
ax0.plot(range(1,100),RMSE_train,label='Train_Data')
ax0.legend()
ax0.set_title('RMSE with respect to the max depth')
ax0.set_xlabel('max_depth')
ax0.set_ylabel('RMSE')
plt.show()

#We can see the RMSE will drop significantly for training data.
```



Grid Search

In [5]:

```
depth = range(1,20)
clf = DecisionTreeRegressor(criterion="mse", max_depth=depth)
parameters={'max_depth': range(1,20,2)}
grid = GridSearchCV(clf, parameters,cv = 10, scoring = 'r2')
grid.fit(X_train,y_train)
```

Out[5]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=DecisionTreeRegressor(criterion='mse',
                                             max_depth=range(1, 20),
                                             max_features=None,
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             presort=False, random_state=None,
                                             splitter='best'),
             iid='warn', n_jobs=None, param_grid={'max_depth': range(1, 20, 2)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='r2', verbose=0)
```

In [6]:

```
print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("R Squared: ",r2_score(y_test, grid.predict(X_test)))
print('RMSE:', np.sqrt(np.sum(((y_test-grid.predict(X_test))**2)/len(y_test))))

#depth is 3, we have r2 score of 0.575 and RMSE of 112
```

```
0.4024270151925781
{'max_depth': 3}
DecisionTreeRegressor(criterion='mse', max_depth=3, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
R Squared:  0.57506404622763
RMSE: 112.07941506787904
```

KNN

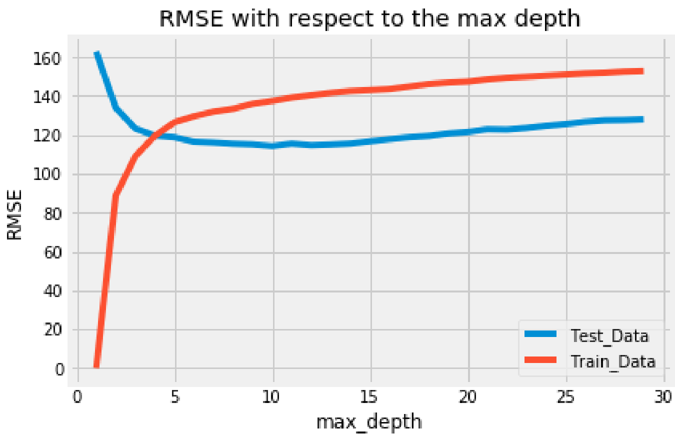
In [26]:

```
from sklearn.preprocessing import StandardScaler
```

```python
from sklearn import neighbors, datasets
scaler=StandardScaler()
```

In [8]:

```python
scaler.fit(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
fig = plt.figure()
ax0 = fig.add_subplot(111)
RMSE_train = []
RMSE_test = []
for i in range(1,30):
    #Paramterize the model and let i be the number of minimum instances per leaf node
    knn = neighbors.KNeighborsRegressor(n_neighbors=i)
    #Train the model
    knn.fit(X_train,y_train)
    #Predict query instances
    predicted_train = knn.predict(X_train)
    predicted_test = knn.predict(X_test)
    #Calculate and append the RMSEs
    RMSE_train.append(np.sqrt(np.sum(((y_train-predicted_train)**2)/len(y_train))))
    RMSE_test.append(np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))

ax0.plot(range(1,30),RMSE_test,label='Test_Data')
ax0.plot(range(1,30),RMSE_train,label='Train_Data')
ax0.legend()
ax0.set_title('RMSE with respect to the max depth')
ax0.set_xlabel('max_depth')
ax0.set_ylabel('RMSE')
plt.show()

#We scale our data first then do the analysis.
#There is no overfitting issue here unless we choose a really small K
```



Grid Search

In [9]:

```python
k_range = list(range(1,31))
weight_options = ["uniform", "distance"]
param_grid = dict(n_neighbors = k_range, weights = weight_options)
knn = neighbors.KNeighborsRegressor()
grid = GridSearchCV(knn, param_grid, cv = 10, scoring = 'r2')
grid.fit(X_train,y_train)
```

Out[9]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30,
                                           metric='minkowski',
                                           metric_params=None, n_jobs=None,
                                           n_neighbors=5, p=2,
                                           weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
```

```
param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                            13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
                            23, 24, 25, 26, 27, 28, 29, 30],
             'weights': ['uniform', 'distance']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
       scoring='r2', verbose=0)
```

In [10]:

```
print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Prediction Accuracy: ",r2_score(y_test, grid.predict(X_test)))
```

```
0.3521136522247128
{'n_neighbors': 11, 'weights': 'distance'}
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=11, p=2,
                    weights='distance')
Prediction Accuracy:  0.5579974545654907
```

In [11]:

```
knn=neighbors.KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=11, p=2,
                    weights='distance')
knn.fit(X_train,y_train)
predicted_test = grid.predict(X_test)

print('RMSE:', np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))
print("R Squared: ",r2_score(y_test, predicted_test))

#We choose K to be 11.
#This gives us r2 score of 0.558 and RMSE equals to 114.
```

```
RMSE: 114.30796727731759
R Squared:  0.5579974545654907
```

Linear Regression

In [11]:

```
from sklearn import linear_model
lasso = linear_model.LassoCV()
lasso.fit(X_train, y_train)

predicted_test = lasso.predict(X_test)

print('RMSE:', np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))
print("R Squared: ",r2_score(y_test, predicted_test))

#We use LASSO and we get a 0.648 r2 amd RMSE equals to 102.
```

```
RMSE: 101.99670965500275
R Squared:  0.6480798851995194
```

```
/Users/yutingxin/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

SVR

In [14]:

```
from sklearn.svm import SVR
```

In [14]:

```
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

scores = ['r2']

for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(SVR(), tuned_parameters, cv=5
                       )
    clf.fit(X_train, y_train)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    print()
```

```
# Tuning hyper-parameters for r2

Best parameters set found on development set:

{'C': 1000, 'kernel': 'linear'}

Grid scores on development set:
```

In [15]:

```
clf=SVR(kernel='linear', C=1000)
clf.fit(X_train, y_train)

predicted_test = clf.predict(X_test)

print('RMSE:', np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))
print("R Squared: ",r2_score(y_test, predicted_test))
```

```
RMSE: 111.7602493144994
R Squared:  0.5774807593372118
```

Neural Network

In [2]:

```
import keras.backend as K
from keras.models import Sequential
from keras.datasets import mnist
from keras.layers import Dense
from keras.utils import np_utils
from keras.wrappers.scikit_learn import KerasRegressor
from kerastuner.tuners import RandomSearch
from tensorflow.keras import layers
from tensorflow import keras


from kerastuner.engine.hypermodel import HyperModel
from kerastuner.engine.hyperparameters import HyperParameters
```

```
Using TensorFlow backend.
```

Tune layers, batch size, and epochs

In [36]:

```
def create_model(hidden_layers=1):
  # Initialize the constructor
  model = Sequential()
  # Add an input layer
  model.add(Dense(23, activation='relu', input_dim=23))

  for i in range(hidden_layers):
      # Add one hidden layer
      model.add(Dense(15, activation='relu'))

  # Add an output layer
  model.add(Dense(1, activation='relu'))
  #compile model
  model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
  return model
```

In [37]:

```
model=KerasRegressor(build_fn=create_model,batch_size=1000,epochs=10)
```

In [40]:

```
epochs=[1,10,100]
batch_size=[5,10,50,100]
hidden_layers=[2,3,4,5,6,7,8]
param_grid=dict(epochs=epochs,batch_size=batch_size,hidden_layers=hidden_layers)
```

In [41]:

```
grid=GridSearchCV(estimator=model,param_grid=param_grid,n_jobs=-1,cv=3)
grid_result=grid.fit(X_train,y_train)
```

```
C:\Games\anaconoda\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: T
he default of the `iid` parameter will change from True to False in version 0.22 and will be
removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

```
Epoch 1/10
1600/1600 [==============================] - 0s 131us/step - loss: 45444.3271 - accuracy: 0.0694
Epoch 2/10
1600/1600 [==============================] - 0s 67us/step - loss: 29378.2458 - accuracy: 0.0012
Epoch 3/10
1600/1600 [==============================] - 0s 66us/step - loss: 23196.2610 - accuracy: 6.2500e-0
4
Epoch 4/10
1600/1600 [==============================] - 0s 70us/step - loss: 19671.5845 - accuracy: 0.0012
Epoch 5/10
1600/1600 [==============================] - 0s 72us/step - loss: 18099.0144 - accuracy: 0.0125
Epoch 6/10
1600/1600 [==============================] - 0s 72us/step - loss: 17373.1946 - accuracy: 0.0237
Epoch 7/10
1600/1600 [==============================] - 0s 72us/step - loss: 17081.6632 - accuracy: 0.0450
Epoch 8/10
1600/1600 [==============================] - 0s 73us/step - loss: 16916.1921 - accuracy: 0.0331
Epoch 9/10
1600/1600 [==============================] - 0s 70us/step - loss: 16701.3390 - accuracy: 0.0419
Epoch 10/10
1600/1600 [==============================] - 0s 68us/step - loss: 16651.5940 - accuracy: 0.0431
```

In [42]:

```
grid.best_params_
#hidden layers=3, epochs = 10, batch size=10
```

Out[42]:

```
{'batch_size': 10, 'epochs': 10, 'hidden_layers': 3}
```

In [17]:

```
def create_model():

    model = Sequential()
    model.add(Dense(23, input_dim=23, activation='relu'))
    model.add(Dense(1, activation='relu'))
    model.compile(loss='mean_squared_error', optimizer='adam',metrics=['accuracy'])
    return model
model=KerasRegressor(build_fn=create_model,batch_size=1000,epochs=10)
```

In [18]:

```
epochs=[1,10,50,100]
batch_size=[5,10,50,100,1000]

param_grid=dict(epochs=epochs,batch_size=batch_size)
grid=GridSearchCV(estimator=model,param_grid=param_grid,n_jobs=-1,cv=3)
grid_result=grid.fit(X_train,y_train)
```

```
C:\Games\anaconoda\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: T
he default of the `iid` parameter will change from True to False in version 0.22 and will be
removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

```
Epoch 1/100
1600/1600 [==============================] - 0s 139us/step - loss: 46193.5515 - accuracy: 0.0519
Epoch 2/100
1600/1600 [==============================] - 0s 106us/step - loss: 44531.1314 - accuracy: 0.0150
Epoch 3/100
1600/1600 [==============================] - 0s 109us/step - loss: 41605.1618 - accuracy: 0.0037
Epoch 4/100
1600/1600 [==============================] - 0s 108us/step - loss: 37999.2390 - accuracy:
0.0000e+00
Epoch 5/100
1600/1600 [==============================] - 0s 112us/step - loss: 34402.8988 - accuracy:
0.0000e+00
Epoch 6/100
1600/1600 [==============================] - 0s 111us/step - loss: 31195.0772 - accuracy: 0.0019
Epoch 7/100
1600/1600 [==============================] - 0s 110us/step - loss: 28592.4038 - accuracy: 6.2500e-
04
Epoch 8/100
1600/1600 [==============================] - 0s 111us/step - loss: 26553.2462 - accuracy: 6.2500e-
04
Epoch 9/100
1600/1600 [==============================] - 0s 110us/step - loss: 24940.3545 - accuracy:
0.0000e+00
Epoch 10/100
1600/1600 [==============================] - 0s 108us/step - loss: 23702.6493 - accuracy: 0.0012
Epoch 11/100
1600/1600 [==============================] - 0s 110us/step - loss: 22717.3152 - accuracy: 6.2500e-
04
Epoch 12/100
1600/1600 [==============================] - 0s 112us/step - loss: 21901.0448 - accuracy:
0.0000e+00
Epoch 13/100
1600/1600 [==============================] - 0s 106us/step - loss: 21203.8680 - accuracy: 6.2500e-
04
Epoch 14/100
1600/1600 [==============================] - 0s 107us/step - loss: 20569.3808 - accuracy: 0.0019
Epoch 15/100
1600/1600 [==============================] - 0s 110us/step - loss: 20008.3742 - accuracy: 0.0025
Epoch 16/100
1600/1600 [==============================] - 0s 110us/step - loss: 19534.5956 - accuracy: 0.0012
Epoch 17/100
1600/1600 [==============================] - 0s 110us/step - loss: 19123.2991 - accuracy: 0.0012
Epoch 18/100
1600/1600 [==============================] - 0s 109us/step - loss: 18772.6280 - accuracy: 0.0019
Epoch 19/100
1600/1600 [==============================] - 0s 115us/step - loss: 18473.8976 - accuracy: 0.0025
Epoch 20/100
1600/1600 [==============================] - 0s 120us/step - loss: 18222.8724 - accuracy: 0.0063
Epoch 21/100
1600/1600 [==============================] - 0s 110us/step - loss: 18008.0155 - accuracy: 0.0069
Epoch 22/100
1600/1600 [==============================] - 0s 110us/step - loss: 17831.5274 - accuracy: 0.0144
```

```
Epoch 23/100
1600/1600 [==============================] - 0s 110us/step - loss: 17671.0547 - accuracy: 0.0194
Epoch 24/100
1600/1600 [==============================] - 0s 111us/step - loss: 17552.7746 - accuracy: 0.0275
Epoch 25/100
1600/1600 [==============================] - 0s 111us/step - loss: 17440.2251 - accuracy: 0.0325
Epoch 26/100
1600/1600 [==============================] - 0s 109us/step - loss: 17349.5556 - accuracy: 0.0394
Epoch 27/100
1600/1600 [==============================] - 0s 110us/step - loss: 17278.8063 - accuracy: 0.0450
Epoch 28/100
1600/1600 [==============================] - 0s 110us/step - loss: 17225.6953 - accuracy: 0.0500
Epoch 29/100
1600/1600 [==============================] - 0s 112us/step - loss: 17152.3474 - accuracy: 0.0569
Epoch 30/100
1600/1600 [==============================] - 0s 111us/step - loss: 17088.0568 - accuracy: 0.0600
Epoch 31/100
1600/1600 [==============================] - 0s 111us/step - loss: 17044.2289 - accuracy: 0.0625
Epoch 32/100
1600/1600 [==============================] - 0s 111us/step - loss: 16991.6907 - accuracy: 0.0675
Epoch 33/100
1600/1600 [==============================] - 0s 111us/step - loss: 16958.0874 - accuracy: 0.0675
Epoch 34/100
1600/1600 [==============================] - 0s 113us/step - loss: 16920.5600 - accuracy: 0.0700
Epoch 35/100
1600/1600 [==============================] - 0s 112us/step - loss: 16887.4571 - accuracy: 0.0688
Epoch 36/100
1600/1600 [==============================] - 0s 111us/step - loss: 16853.0153 - accuracy: 0.0737
Epoch 37/100
1600/1600 [==============================] - 0s 107us/step - loss: 16821.6401 - accuracy: 0.0725
Epoch 38/100
1600/1600 [==============================] - 0s 107us/step - loss: 16794.1619 - accuracy: 0.0744
Epoch 39/100
1600/1600 [==============================] - 0s 109us/step - loss: 16765.0332 - accuracy: 0.0787
Epoch 40/100
1600/1600 [==============================] - 0s 112us/step - loss: 16736.3295 - accuracy: 0.0775
Epoch 41/100
1600/1600 [==============================] - 0s 107us/step - loss: 16709.0598 - accuracy: 0.0800
Epoch 42/100
1600/1600 [==============================] - 0s 106us/step - loss: 16687.6429 - accuracy: 0.0800
Epoch 43/100
1600/1600 [==============================] - 0s 109us/step - loss: 16654.6796 - accuracy: 0.0819
Epoch 44/100
1600/1600 [==============================] - 0s 109us/step - loss: 16644.8950 - accuracy: 0.0831
Epoch 45/100
1600/1600 [==============================] - 0s 113us/step - loss: 16615.3525 - accuracy: 0.0825
Epoch 46/100
1600/1600 [==============================] - ETA: 0s - loss: 16180.1782 - accuracy: 0.08 - 0s 107u
s/step - loss: 16604.4598 - accuracy: 0.0869
Epoch 47/100
1600/1600 [==============================] - ETA: 0s - loss: 15663.4423 - accuracy: 0.08 - 0s 110u
s/step - loss: 16581.2077 - accuracy: 0.0862
Epoch 48/100
1600/1600 [==============================] - 0s 111us/step - loss: 16551.8044 - accuracy: 0.0844
Epoch 49/100
1600/1600 [==============================] - 0s 115us/step - loss: 16534.7384 - accuracy: 0.0862
Epoch 50/100
1600/1600 [==============================] - 0s 121us/step - loss: 16512.5977 - accuracy: 0.0900
Epoch 51/100
1600/1600 [==============================] - 0s 110us/step - loss: 16488.6343 - accuracy: 0.0900
Epoch 52/100
1600/1600 [==============================] - 0s 110us/step - loss: 16476.8082 - accuracy: 0.0887
Epoch 53/100
1600/1600 [==============================] - 0s 112us/step - loss: 16449.9052 - accuracy: 0.0906
Epoch 54/100
1600/1600 [==============================] - 0s 113us/step - loss: 16435.4371 - accuracy: 0.0862
Epoch 55/100
1600/1600 [==============================] - 0s 112us/step - loss: 16419.7672 - accuracy: 0.0894
Epoch 56/100
1600/1600 [==============================] - 0s 112us/step - loss: 16402.5156 - accuracy: 0.0894
Epoch 57/100
1600/1600 [==============================] - 0s 110us/step - loss: 16385.1231 - accuracy: 0.0925
Epoch 58/100
1600/1600 [==============================] - 0s 112us/step - loss: 16372.0957 - accuracy: 0.0925
Epoch 59/100
1600/1600 [==============================] - 0s 110us/step - loss: 16349.5315 - accuracy: 0.0919
Epoch 60/100
```

```
1600/1600 [==============================] - 0s 111us/step - loss: 16334.1156 - accuracy: 0.0944
Epoch 61/100
1600/1600 [==============================] - 0s 111us/step - loss: 16326.1557 - accuracy: 0.0956
Epoch 62/100
1600/1600 [==============================] - 0s 112us/step - loss: 16297.7565 - accuracy: 0.0962
Epoch 63/100
1600/1600 [==============================] - 0s 111us/step - loss: 16286.4741 - accuracy: 0.0931
Epoch 64/100
1600/1600 [==============================] - 0s 112us/step - loss: 16259.8506 - accuracy: 0.0950
Epoch 65/100
1600/1600 [==============================] - 0s 114us/step - loss: 16251.5810 - accuracy: 0.0944
Epoch 66/100
1600/1600 [==============================] - 0s 112us/step - loss: 16232.0848 - accuracy: 0.0944
Epoch 67/100
1600/1600 [==============================] - 0s 113us/step - loss: 16216.7838 - accuracy: 0.0938
Epoch 68/100
1600/1600 [==============================] - 0s 113us/step - loss: 16208.9213 - accuracy: 0.0975
Epoch 69/100
1600/1600 [==============================] - 0s 112us/step - loss: 16189.4831 - accuracy: 0.0994
Epoch 70/100
1600/1600 [==============================] - 0s 113us/step - loss: 16173.4083 - accuracy: 0.0969
Epoch 71/100
1600/1600 [==============================] - 0s 112us/step - loss: 16149.3267 - accuracy: 0.0994
Epoch 72/100
1600/1600 [==============================] - 0s 111us/step - loss: 16141.1936 - accuracy: 0.0994
Epoch 73/100
1600/1600 [==============================] - 0s 107us/step - loss: 16123.5342 - accuracy: 0.1000
Epoch 74/100
1600/1600 [==============================] - 0s 110us/step - loss: 16108.4594 - accuracy: 0.1000
Epoch 75/100
1600/1600 [==============================] - 0s 108us/step - loss: 16106.1901 - accuracy: 0.1013
Epoch 76/100
1600/1600 [==============================] - 0s 110us/step - loss: 16079.0504 - accuracy: 0.1019
Epoch 77/100
1600/1600 [==============================] - 0s 111us/step - loss: 16079.3763 - accuracy: 0.1006
Epoch 78/100
1600/1600 [==============================] - 0s 114us/step - loss: 16060.0377 - accuracy: 0.1050
Epoch 79/100
1600/1600 [==============================] - 0s 121us/step - loss: 16041.9655 - accuracy: 0.1044
Epoch 80/100
1600/1600 [==============================] - 0s 112us/step - loss: 16025.6414 - accuracy: 0.1044
Epoch 81/100
1600/1600 [==============================] - 0s 111us/step - loss: 16019.5942 - accuracy: 0.1069
Epoch 82/100
1600/1600 [==============================] - 0s 112us/step - loss: 16002.1213 - accuracy: 0.1094
Epoch 83/100
1600/1600 [==============================] - 0s 112us/step - loss: 15993.7208 - accuracy: 0.1075
Epoch 84/100
1600/1600 [==============================] - 0s 109us/step - loss: 15973.6291 - accuracy: 0.1056
Epoch 85/100
1600/1600 [==============================] - 0s 110us/step - loss: 15965.9703 - accuracy: 0.1100
Epoch 86/100
1600/1600 [==============================] - 0s 112us/step - loss: 15947.0591 - accuracy: 0.1119
Epoch 87/100
1600/1600 [==============================] - 0s 108us/step - loss: 15930.7256 - accuracy: 0.1144
Epoch 88/100
1600/1600 [==============================] - 0s 105us/step - loss: 15921.1697 - accuracy: 0.1169
Epoch 89/100
1600/1600 [==============================] - 0s 107us/step - loss: 15908.8949 - accuracy: 0.1163
Epoch 90/100
1600/1600 [==============================] - 0s 106us/step - loss: 15895.6096 - accuracy: 0.1163
Epoch 91/100
1600/1600 [==============================] - 0s 108us/step - loss: 15881.8782 - accuracy: 0.1213
Epoch 92/100
1600/1600 [==============================] - 0s 110us/step - loss: 15875.3411 - accuracy: 0.1231
Epoch 93/100
1600/1600 [==============================] - 0s 108us/step - loss: 15856.9114 - accuracy: 0.1256
Epoch 94/100
1600/1600 [==============================] - 0s 110us/step - loss: 15841.7659 - accuracy: 0.1250
Epoch 95/100
1600/1600 [==============================] - 0s 112us/step - loss: 15828.5195 - accuracy: 0.1287
Epoch 96/100
1600/1600 [==============================] - 0s 111us/step - loss: 15811.9951 - accuracy: 0.1325
Epoch 97/100
1600/1600 [==============================] - 0s 112us/step - loss: 15809.5510 - accuracy: 0.1312
Epoch 98/100
1600/1600 [==============================] - 0s 110us/step - loss: 15788.6516 - accuracy: 0.1300
```

```
Epoch 99/100
1600/1600 [==============================] - 0s 110us/step - loss: 15774.8197 - accuracy: 0.1300
Epoch 100/100
1600/1600 [==============================] - 0s 110us/step - loss: 15769.2022 - accuracy: 0.1319
```

In [19]:

```
grid.best_params_
```

Out[19]:

```
{'batch_size': 5, 'epochs': 100}
```

Tune Optimizer

In [48]:

```
def create_model(hidden_layers=3,optimizer='adam'):
  # Initialize the constructor
  model = Sequential()
  # Add an input layer
  model.add(Dense(23, activation='relu', input_dim=23))

  for i in range(hidden_layers):
      # Add one hidden layer
      model.add(Dense(15, activation='relu'))

  # Add an output layer
  model.add(Dense(1, activation='relu'))
  #compile model
  model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['accuracy'])
  return model
```

In [49]:

```
model=KerasRegressor(build_fn=create_model,batch_size=10,epochs=10)
```

In [50]:

```
optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax']
param_grid = dict(optimizer=optimizer)
grid=GridSearchCV(estimator=model,param_grid=param_grid,n_jobs=-1,cv=3)
grid_result=grid.fit(X_train,y_train)
```

```
C:\Games\anaconoda\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: T
he default of the `iid` parameter will change from True to False in version 0.22 and will be
removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

```
Epoch 1/10
1600/1600 [==============================] - 0s 127us/step - loss: 33892.1772 - accuracy: 0.0338
Epoch 2/10
1600/1600 [==============================] - 0s 71us/step - loss: 18406.6705 - accuracy:
0.0000e+00
Epoch 3/10
1600/1600 [==============================] - 0s 72us/step - loss: 16845.2138 - accuracy:
0.0000e+00
Epoch 4/10
1600/1600 [==============================] - 0s 72us/step - loss: 16209.8780 - accuracy: 6.2500e-0
4
Epoch 5/10
1600/1600 [==============================] - 0s 73us/step - loss: 15954.7343 - accuracy:
0.0000e+00
Epoch 6/10
1600/1600 [==============================] - 0s 72us/step - loss: 15751.5146 - accuracy: 6.2500e-0
4
Epoch 7/10
1600/1600 [==============================] - 0s 72us/step - loss: 15742.8539 - accuracy:
0.0000e+00
Epoch 8/10
```

```
1600/1600 [==============================] - 0s 64us/step - loss: 15649.8924 - accuracy: 6.2500e-0
4
Epoch 9/10
1600/1600 [==============================] - 0s 67us/step - loss: 15538.7462 - accuracy: 0.0025
Epoch 10/10
1600/1600 [==============================] - 0s 67us/step - loss: 15486.2765 - accuracy: 0.0031
```

In [53]:

```
grid.best_params_
#adadelta is the best optimizer
```

Out[53]:

```
{'optimizer': 'Adadelta'}
```

Tune learning rate and momentum

In [68]:

```
def create_model(hidden_layers=3,learn_rate=0.01, momentum=0):
  # Initialize the constructor
  model = Sequential()
  # Add an input layer
  model.add(Dense(23, activation='relu', input_dim=23))

  for i in range(hidden_layers):
      # Add one hidden layer
      model.add(Dense(15, activation='relu'))

  # Add an output layer
  optimizer = SGD(lr=learn_rate, momentum=momentum)
  model.add(Dense(1, activation='relu'))
  #compile model
  model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['accuracy'])
  return model
```

In [69]:

```
model=KerasRegressor(build_fn=create_model,batch_size=10,epochs=10)
learn_rate = [0.001, 0.01, 0.1, 0.2, 0.3]
momentum = [0.0, 0.2, 0.4, 0.6, 0.8, 0.9]
param_grid = dict(learn_rate=learn_rate, momentum=momentum)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train, y_train)
```

```
C:\Games\anaconoda\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: T
he default of the `iid` parameter will change from True to False in version 0.22 and will be
removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

```
Epoch 1/10
1600/1600 [==============================] - 0s 88us/step - loss: 46775.8847 - accuracy: 0.2837
Epoch 2/10
1600/1600 [==============================] - 0s 57us/step - loss: 46775.8846 - accuracy: 0.2837
Epoch 3/10
1600/1600 [==============================] - 0s 60us/step - loss: 46775.8851 - accuracy: 0.2837
Epoch 4/10
1600/1600 [==============================] - 0s 60us/step - loss: 46775.8849 - accuracy: 0.2837
Epoch 5/10
1600/1600 [==============================] - 0s 59us/step - loss: 46775.8851 - accuracy: 0.2837
Epoch 6/10
1600/1600 [==============================] - 0s 59us/step - loss: 46775.8852 - accuracy: 0.2837
Epoch 7/10
1600/1600 [==============================] - 0s 61us/step - loss: 46775.8850 - accuracy: 0.2837
Epoch 8/10
1600/1600 [==============================] - 0s 60us/step - loss: 46775.8852 - accuracy: 0.2837
Epoch 9/10
1600/1600 [==============================] - 0s 59us/step - loss: 46775.8852 - accuracy: 0.2837
Epoch 10/10
1600/1600 [==============================] - 0s 59us/step - loss: 46775.8856 - accuracy: 0.2837
```

In [70]:

```
grid.best_params_
```

Out[70]:

```
{'learn_rate': 0.001, 'momentum': 0.0}
```

Tune activation function

In [76]:

```
def create_model(hidden_layers=3,learn_rate=0.001, momentum=0,activation='relu'):
  # Initialize the constructor
  model = Sequential()
  # Add an input layer
  model.add(Dense(23, activation=activation, input_dim=23))

  for i in range(hidden_layers):
      # Add one hidden layer
      model.add(Dense(15, activation=activation))

  # Add an output layer
  model.add(Dense(1, activation='relu'))
  #compile model
  model.compile(loss='mean_squared_error', optimizer='Adadelta', metrics=['accuracy'])
  return model
```

In [77]:

```
model=KerasRegressor(build_fn=create_model,batch_size=10,epochs=10)
activation=['relu','sigmoid','linear','hard_sigmoid']
param_grid = dict(activation=activation)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train, y_train)
```

```
C:\Games\anaconoda\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: T
he default of the `iid` parameter will change from True to False in version 0.22 and will be
removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

```
Epoch 1/10
1600/1600 [==============================] - 0s 133us/step - loss: 43913.2771 - accuracy: 0.1912
Epoch 2/10
1600/1600 [==============================] - 0s 71us/step - loss: 20422.6059 - accuracy:
0.0000e+00
Epoch 3/10
1600/1600 [==============================] - 0s 71us/step - loss: 17548.1392 - accuracy: 0.0044
Epoch 4/10
1600/1600 [==============================] - 0s 70us/step - loss: 16400.6477 - accuracy: 0.0325
Epoch 5/10
1600/1600 [==============================] - 0s 69us/step - loss: 16408.2446 - accuracy: 0.0562
Epoch 6/10
1600/1600 [==============================] - 0s 69us/step - loss: 16082.3753 - accuracy: 0.0631
Epoch 7/10
1600/1600 [==============================] - 0s 71us/step - loss: 16067.8893 - accuracy: 0.0694
Epoch 8/10
1600/1600 [==============================] - 0s 72us/step - loss: 15973.7011 - accuracy: 0.0781
Epoch 9/10
1600/1600 [==============================] - 0s 77us/step - loss: 15912.7157 - accuracy: 0.0775
Epoch 10/10
1600/1600 [==============================] - 0s 72us/step - loss: 15864.8151 - accuracy: 0.0825
```

In [78]:

```
grid.best_params_
#best activation function is 'relu'
```

Out[78]:

Out[76]:

```
{'activation': 'relu'}
```

Tune number of neurons

In [85]:

```
def create_model(hidden_layers=3,learn_rate=0.001, momentum=0,neurons=1):
  # Initialize the constructor
  model = Sequential()
  # Add an input layer
  model.add(Dense(23, activation='relu', input_dim=23))

  for i in range(hidden_layers):
      # Add one hidden layer
      model.add(Dense(neurons, activation='relu'))

  # Add an output layer
  model.add(Dense(1, activation='relu'))
  #compile model
  model.compile(loss='mean_squared_error', optimizer='Adadelta', metrics=['accuracy'])
  return model
```

In [86]:

```
model=KerasRegressor(build_fn=create_model,batch_size=10,epochs=10)
neurons=[1,5,10,15,20]
param_grid = dict(neurons=neurons)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train, y_train)
```

```
C:\Games\anaconoda\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: T
he default of the `iid` parameter will change from True to False in version 0.22 and will be
removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

```
Epoch 1/10
1600/1600 [==============================] - 0s 132us/step - loss: 33916.4041 - accuracy: 0.0131
Epoch 2/10
1600/1600 [==============================] - 0s 74us/step - loss: 18669.7615 - accuracy: 6.2500e-0
4
Epoch 3/10
1600/1600 [==============================] - 0s 72us/step - loss: 16958.5383 - accuracy:
0.0000e+00
Epoch 4/10
1600/1600 [==============================] - 0s 71us/step - loss: 16309.1429 - accuracy: 6.2500e-0
4
Epoch 5/10
1600/1600 [==============================] - 0s 74us/step - loss: 16159.4663 - accuracy:
0.0000e+00
Epoch 6/10
1600/1600 [==============================] - 0s 76us/step - loss: 15956.7268 - accuracy:
0.0000e+00
Epoch 7/10
1600/1600 [==============================] - 0s 74us/step - loss: 15848.7987 - accuracy: 6.2500e-0
4
Epoch 8/10
1600/1600 [==============================] - 0s 72us/step - loss: 15793.2902 - accuracy:
0.0000e+00
Epoch 9/10
1600/1600 [==============================] - 0s 73us/step - loss: 15711.6494 - accuracy:
0.0000e+00
Epoch 10/10
1600/1600 [==============================] - 0s 69us/step - loss: 15633.7778 - accuracy: 0.0019
```

In [87]:

```
grid.best_params_
```

Out[87]:

```
{'neurons': 20}
```

Prediction

In [394]:

```
def create_model(hidden_layers=3,learn_rate=0.001, momentum=0):
  # Initialize the constructor
  model = Sequential()
  # Add an input layer
  model.add(Dense(23, activation='relu', input_dim=23))

  for i in range(hidden_layers):
      # Add one hidden layer
      model.add(Dense(20, activation='relu'))

  # Add an output layer
  model.add(Dense(1, activation='relu'))
  #compile model
  model.compile(loss='mean_squared_error', optimizer='Adadelta', metrics=['accuracy'])
  return model
model=KerasRegressor(build_fn=create_model,batch_size=10,epochs=10)
```

In [90]:

```
model.fit(X_train, y_train)
predicted_test = model.predict(X_test)

print('RMSE:', np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))
print("R Squared: ",r2_score(y_test, predicted_test))
```

```
Epoch 1/10
1600/1600 [==============================] - 0s 137us/step - loss: 29424.3670 - accuracy: 0.0194
Epoch 2/10
1600/1600 [==============================] - 0s 69us/step - loss: 17892.1291 - accuracy: 6.2500e-0
4
Epoch 3/10
1600/1600 [==============================] - 0s 70us/step - loss: 16893.3898 - accuracy:
0.0000e+00
Epoch 4/10
1600/1600 [==============================] - 0s 67us/step - loss: 16233.0785 - accuracy:
0.0000e+00
Epoch 5/10
1600/1600 [==============================] - 0s 67us/step - loss: 16055.0469 - accuracy:
0.0000e+00
Epoch 6/10
1600/1600 [==============================] - 0s 71us/step - loss: 15956.9392 - accuracy: 0.0019
Epoch 7/10
1600/1600 [==============================] - 0s 70us/step - loss: 15809.8914 - accuracy: 0.0056
Epoch 8/10
1600/1600 [==============================] - 0s 71us/step - loss: 15777.9572 - accuracy: 0.0081
Epoch 9/10
1600/1600 [==============================] - 0s 71us/step - loss: 15637.0596 - accuracy: 0.0081
Epoch 10/10
1600/1600 [==============================] - 0s 73us/step - loss: 15676.0558 - accuracy: 0.0131
RMSE: 94.78260072825223
R Squared:  0.6961011833968478
```

In [91]:

```
#We get RMSE equals to 94.78 and R squared of 0.696
```

XGBOOST

In [93]:

```
from xgboost.sklearn import XGBRegressor
import warnings; warnings.simplefilter('ignore')
```

In [94]:

```
from xgboost.sklearn import XGBRegressor
import scipy.stats as st

one_to_left = st.beta(10, 1)
from_zero_positive = st.expon(0, 50)

params = {
    "n_estimators": st.randint(3, 40),
    "max_depth": st.randint(3, 40),
    "learning_rate": st.uniform(0.05, 0.4),
    "colsample_bytree": one_to_left,
    "subsample": one_to_left,
    "gamma": st.uniform(0, 10),
    'reg_alpha': from_zero_positive,
    "min_child_weight": from_zero_positive,
}

xgbreg = XGBRegressor(nthreads=-1)
```

In [95]:

```
from sklearn.model_selection import RandomizedSearchCV

gs = RandomizedSearchCV(xgbreg, params, n_jobs=1)
gs.fit(X_train, y_train)
```

```
[18:21:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
```

```
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[18:21:43] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[95]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, nthreads=-1, objective='reg:linear',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=None, subsample=1, verbosity=1),
          fit_params=None, iid='warn', n_iter=10, n_jobs=1,
          param_distributions={'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object
at 0x000001EA9C03A160>, 'max_depth': <scipy.stats._distn_infrastructure.rv_frozen object at
0x000001EA9C028828>, 'learning_rate': <scipy.stats._distn_infrastructure.rv_frozen object at 0x000
001EA9C012E10>, 'cols...98>, 'min_child_weight': <scipy.stats._distn_infrastructure.rv_frozen obje
ct at 0x000001EA9C03AF98>},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=0)
```

In [96]:

```
gs.best_estimator_
```

Out[96]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.8829720261487249,
       gamma=4.300202312684424, importance_type='gain',
       learning_rate=0.31855410235386916, max_delta_step=0, max_depth=18,
       min_child_weight=42.00033752829647, missing=None, n_estimators=20,
       n_jobs=1, nthread=None, nthreads=-1, objective='reg:linear',
       random_state=0, reg_alpha=211.14816580068515, reg_lambda=1,
       scale_pos_weight=1, seed=None, silent=None,
       subsample=0.9664457525643635, verbosity=1)
```

In [97]:

```
predicted_test = gs.predict(X_test)

print('RMSE:', np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))
print("R Squared: ",r2_score(y_test, predicted_test))
```

```
RMSE: 105.45905695923733
R Squared:  0.6237820317999885
```

Conclusion

From previous model building, we can see that neural network gives us the best model to predict spendings. It has the lowest RMSE among all models, which is 94.78 and best r squared score, which is 0.696.

Problem 1B

Prepare data

In [57]:

```
data1=data[data.Purchase==1]
```

In [58]:

```
data1.head()
```

Out[58]:

	sequence_number	US	source_a	source_c	source_b	source_d	source_e	source_m	source_o	source_h	...	source_x	source_
0	1	1	0	0	1	0	0	0	0	0	...	0	
2	3	1	0	0	0	0	0	0	0	0	...	0	
8	9	1	1	0	0	0	0	0	0	0	...	0	
9	10	1	1	0	0	0	0	0	0	0	...	0	
13	14	1	1	0	0	0	0	0	0	0	...	0	

5 rows × 25 columns

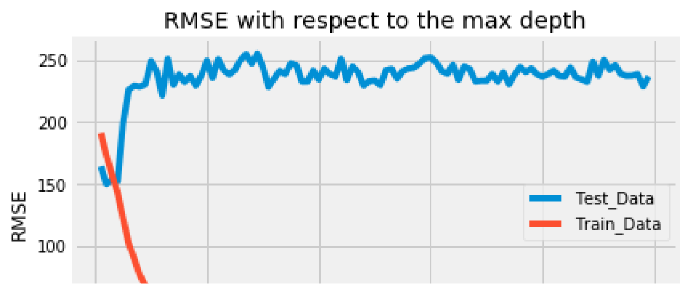
In [59]:

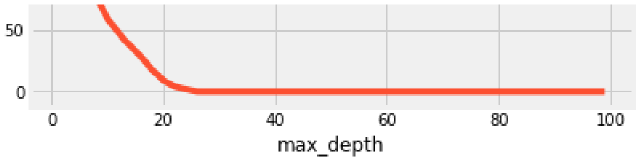
```
X=data1.iloc[:,0:-2]
y=data1.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=100)
```

Regression Tree

In [36]:

```
fig = plt.figure()
ax0 = fig.add_subplot(111)
RMSE_train = []
RMSE_test = []
for i in range(1,100):
    #Paramterize the model and let i be the number of minimum instances per leaf node
    regression_model = DecisionTreeRegressor(criterion="mse",max_depth=i)
    #Train the model
    regression_model.fit(X_train,y_train)
    #Predict query instances
    predicted_train = regression_model.predict(X_train)
    predicted_test = regression_model.predict(X_test)
    #Calculate and append the RMSEs
    RMSE_train.append(np.sqrt(np.sum(((y_train-predicted_train)**2)/len(y_train))))
    RMSE_test.append(np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))

ax0.plot(range(1,100),RMSE_test,label='Test_Data')
ax0.plot(range(1,100),RMSE_train,label='Train_Data')
ax0.legend()
ax0.set_title('RMSE with respect to the max depth')
ax0.set_xlabel('max_depth')
ax0.set_ylabel('RMSE')
plt.show()
```





In [37]:

```
depth = range(1,20)
clf = DecisionTreeRegressor(criterion="mse", max_depth=depth)
parameters={'max_depth': range(1,20,2)}
grid = GridSearchCV(clf, parameters,cv = 10, scoring = 'r2')
grid.fit(X_train,y_train)
```

Out[37]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=DecisionTreeRegressor(criterion='mse', max_depth=range(1, 20),
           max_features=None, max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, presort=False, random_state=None,
           splitter='best'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': range(1, 20, 2)}, pre_dispatch='2*n_jobs',
       refit=True, return_train_score='warn', scoring='r2', verbose=0)
```

In [38]:

```
print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("R Squared: ",r2_score(y_test, grid.predict(X_test)))
print('RMSE:', np.sqrt(np.sum(((y_test-grid.predict(X_test))**2)/len(y_test))))

#There is an overfitting issue here and we can see R square is negative
```

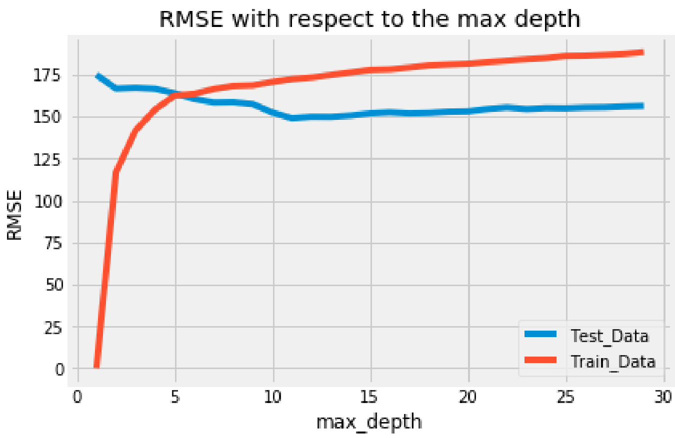
```
0.2925700391569781
{'max_depth': 5}
DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
R Squared:  -0.2170766768597996
RMSE: 199.7098225532082
```

KNN

In [40]:

```
scaler.fit(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
fig = plt.figure()
ax0 = fig.add_subplot(111)
RMSE_train = []
RMSE_test = []
for i in range(1,30):
    #Paramterize the model and let i be the number of minimum instances per leaf node
    knn = neighbors.KNeighborsRegressor(n_neighbors=i)
    #Train the model
    knn.fit(X_train,y_train)
    #Predict query instances
    predicted_train = knn.predict(X_train)
    predicted_test = knn.predict(X_test)
    #Calculate and append the RMSEs
    RMSE_train.append(np.sqrt(np.sum(((y_train-predicted_train)**2)/len(y_train))))
    RMSE_test.append(np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))

ax0.plot(range(1,30),RMSE_test,label='Test Data')
```

```
ax0.plot(range(1,30),RMSE_test,label='Test_Data')
ax0.plot(range(1,30),RMSE_train,label='Train_Data')
ax0.legend()
ax0.set_title('RMSE with respect to the max depth')
ax0.set_xlabel('max_depth')
ax0.set_ylabel('RMSE')
plt.show()
```



In [41]:

```
k_range = list(range(1,31))
weight_options = ["uniform", "distance"]
param_grid = dict(n_neighbors = k_range, weights = weight_options)
knn = neighbors.KNeighborsRegressor()
grid = GridSearchCV(knn, param_grid, cv = 10, scoring = 'r2')
grid.fit(X_train,y_train)
```

Out[41]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
          weights='uniform'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30], 'weights': ['uniform', 'distance']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='r2', verbose=0)
```

In [43]:

```
print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print('RMSE:', np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))
print("R Squared: ",r2_score(y_test, grid.predict(X_test)))
```

```
0.2468631002228364
{'n_neighbors': 15, 'weights': 'distance'}
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=15, p=2,
          weights='distance')
RMSE: 156.37638873269353
Prediction Accuracy:  0.29968131812332544
```

Linear Regression

In [61]:

```
lasso = linear_model.LassoCV()
lasso.fit(X_train, y_train)

predicted_test = lasso.predict(X_test)
```

```
print('RMSE:', np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))
print("Prediction Accuracy: ",r2_score(y_test, predicted_test))
```

```
RMSE: 161.55612809297182
Prediction Accuracy:  0.2035363480359078
```

SVR

In [45]:

```
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

scores = ['r2']

for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(SVR(), tuned_parameters, cv=5
                       )
    clf.fit(X_train, y_train)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    print()
```

```
# Tuning hyper-parameters for r2

Best parameters set found on development set:

{'C': 1000, 'kernel': 'linear'}

Grid scores on development set:
```

In [46]:

```
clf=SVR(kernel='linear', C=1000)
clf.fit(X_train, y_train)

predicted_test = clf.predict(X_test)

print('RMSE:', np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))
print("R Squared: ",r2_score(y_test, predicted_test))
```

```
RMSE: 148.01934016513022
R Squared:  0.3314159430102488
```

Neural Network

Tune batch size and epoch

In [14]:

```
def create_model():
  # Initialize the constructor
  model = Sequential()
  # Add an input layer
  model.add(Dense(23, activation='relu', input_dim=23))

  # Add an output layer
  model.add(Dense(1, activation='relu'))
  #compile model
```

```python
    #compile model
    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
    return model
model=KerasRegressor(build_fn=create_model,batch_size=1000,epochs=10)

epochs=[1,10,50,100]
batch_size=[5,10,50,100,1000]

param_grid=dict(epochs=epochs,batch_size=batch_size)
grid=GridSearchCV(estimator=model,param_grid=param_grid,n_jobs=-1,cv=3)
grid_result=grid.fit(X_train,y_train)
```

```
/Users/yutingxin/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_search.py:813: Dep
recationWarning: The default of the `iid` parameter will change from True to False in version 0.22
and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

```
Epoch 1/100
1600/1600 [==============================] - ETA: 29s - loss: 12114.3574 - accuracy: 0.200 - ETA:
0s - loss: 58528.7644 - accuracy: 0.164 - ETA: 0s - loss: 52707.2879 - accuracy: 0.13 - ETA: 0s -
loss: 53413.6859 - accuracy: 0.11 - ETA: 0s - loss: 52079.7602 - accuracy: 0.09 - ETA: 0s - loss:
49683.4576 - accuracy: 0.07 - ETA: 0s - loss: 47341.9914 - accuracy: 0.06 - 0s 254us/step - loss:
46385.0021 - accuracy: 0.0625
Epoch 2/100
1600/1600 [==============================] - ETA: 0s - loss: 16226.2168 - accuracy: 0.0000e+ - ETA
: 0s - loss: 56388.4665 - accuracy: 0.0000e+ - ETA: 0s - loss: 48550.2201 - accuracy: 0.0000e+ - E
TA: 0s - loss: 43191.1078 - accuracy: 0.0000e+ - ETA: 0s - loss: 42226.9807 - accuracy: 0.0000e+ -
ETA: 0s - loss: 42817.5181 - accuracy: 0.0000e+ - 0s 187us/step - loss: 44649.4834 - accuracy: 0.0
000e+00
Epoch 3/100
1600/1600 [==============================] - ETA: 0s - loss: 16304.4160 - accuracy: 0.0000e+ - ETA
: 0s - loss: 46533.0804 - accuracy: 0.0000e+ - ETA: 0s - loss: 45228.2415 - accuracy: 0.0000e+ - E
TA: 0s - loss: 45883.0101 - accuracy: 0.0000e+ - ETA: 0s - loss: 44750.5835 - accuracy: 0.0000e+ -
ETA: 0s - loss: 42576.7803 - accuracy: 0.0000e+ - 0s 185us/step - loss: 41830.9116 - accuracy: 0.0
000e+00
Epoch 4/100
1600/1600 [==============================] - ETA: 0s - loss: 21117.0410 - accuracy: 0.0000e+ - ETA
: 0s - loss: 49132.9294 - accuracy: 0.0000e+ - ETA: 0s - loss: 53216.5739 - accuracy: 0.0000e+ - E
TA: 0s - loss: 49578.1953 - accuracy: 0.0000e+ - ETA: 0s - loss: 44924.6821 - accuracy: 0.0000e+ -
ETA: 0s - loss: 39471.3583 - accuracy: 0.0000e+ - 0s 177us/step - loss: 38450.9113 - accuracy: 0.0
000e+00
Epoch 5/100
1600/1600 [==============================] - ETA: 0s - loss: 720.9100 - accuracy: 0.0000e+ - ETA:
0s - loss: 31855.0893 - accuracy: 0.0000e+ - ETA: 0s - loss: 24413.9261 - accuracy: 0.0000e+ - ETA
: 0s - loss: 33541.9495 - accuracy: 0.0000e+ - ETA: 0s - loss: 37230.5717 - accuracy: 0.0000e+ - E
TA: 0s - loss: 36465.7860 - accuracy: 0.0000e+ - ETA: 0s - loss: 35484.4877 - accuracy: 0.0000e+ -
0s 197us/step - loss: 34956.8684 - accuracy: 0.0000e+00
Epoch 6/100
1600/1600 [==============================] - ETA: 0s - loss: 44393.3984 - accuracy: 0.0000e+ - ETA
: 0s - loss: 42953.2787 - accuracy: 0.0000e+ - ETA: 0s - loss: 41142.1751 - accuracy: 0.0000e+ - E
TA: 0s - loss: 36794.8778 - accuracy: 0.0000e+ - ETA: 0s - loss: 34064.9410 - accuracy: 0.0000e+ -
ETA: 0s - loss: 32257.9540 - accuracy: 0.0000e+ - 0s 188us/step - loss: 31744.8013 - accuracy: 0.0
000e+00
Epoch 7/100
1600/1600 [==============================] - ETA: 0s - loss: 7332.2891 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 32493.0663 - accuracy: 0.0000e+ - ETA: 0s - loss: 32910.5720 - accuracy: 0.0000e+ - E
TA: 0s - loss: 29479.8834 - accuracy: 0.0000e+ - ETA: 0s - loss: 29685.5917 - accuracy: 9.8039e- -
ETA: 0s - loss: 29325.3393 - accuracy: 8.0321e- - ETA: 0s - loss: 28140.7006 - accuracy: 6.7568e-
- 0s 209us/step - loss: 29007.1583 - accuracy: 6.2500e-04
Epoch 8/100
1600/1600 [==============================] - ETA: 0s - loss: 180279.1094 - accuracy: 0.0000e+0 - E
TA: 0s - loss: 28538.2067 - accuracy: 0.0000e+0 - ETA: 0s - loss: 21559.8725 - accuracy: 0.0000e+
- ETA: 0s - loss: 25994.2616 - accuracy: 0.0000e+ - ETA: 0s - loss: 26637.7863 - accuracy:
0.0000e+ - ETA: 0s - loss: 26962.0699 - accuracy: 0.0000e+ - ETA: 0s - loss: 27815.8064 -
accuracy: 6.6007e- - 0s 202us/step - loss: 26808.0518 - accuracy: 6.2500e-04
Epoch 9/100
1600/1600 [==============================] - ETA: 0s - loss: 13368.9639 - accuracy: 0.0000e+ - ETA
: 0s - loss: 15238.8211 - accuracy: 0.0000e+ - ETA: 0s - loss: 22874.3719 - accuracy: 0.0000e+ - E
TA: 0s - loss: 24449.4453 - accuracy: 0.0000e+ - ETA: 0s - loss: 26450.6793 - accuracy: 0.0000e+ -
ETA: 0s - loss: 24831.3998 - accuracy: 0.0000e+ - ETA: 0s - loss: 25452.2720 - accuracy: 6.6890e-
- 0s 204us/step - loss: 25108.9306 - accuracy: 6.2500e-04
Epoch 10/100
1600/1600 [==============================] - ETA: 0s - loss: 23843.1953 - accuracy: 0.0000e+ - ETA
: 0s - loss: 30416.6171 - accuracy: 0.0000e+ - ETA: 0s - loss: 29410.9240 - accuracy: 0.0000e+ - E
TA: 0s - loss: 25017.5775 - accuracy: 0.0000e+ - ETA: 0s - loss: 26610.0642 - accuracy: 0.0000e+ -
ETA: 0s - loss: 24175.5080 - accuracy: 0.0000e+ - 0s 189us/step - loss: 23771.6966 - accuracy: 0.0
```

```
000e+00
Epoch 11/100
1600/1600 [==============================] - ETA: 0s - loss: 26774.8242 - accuracy: 0.0000e+ - ETA
: 0s - loss: 20545.4182 - accuracy: 0.0000e+ - ETA: 0s - loss: 24100.7217 - accuracy: 0.0000e+ - E
TA: 0s - loss: 21578.6043 - accuracy: 0.0026   - ETA: 0s - loss: 20370.7603 - accuracy: 0.00 -
ETA: 0s - loss: 20488.5846 - accuracy: 0.00 - ETA: 0s - loss: 22347.1175 - accuracy: 0.00 - 0s 192
us/step - loss: 22670.2516 - accuracy: 0.0025
Epoch 12/100
1600/1600 [==============================] - ETA: 0s - loss: 3964.3477 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 19763.8735 - accuracy: 0.0140  - ETA: 0s - loss: 19476.4282 - accuracy: 0.01 - ETA: 0
s - loss: 20746.1903 - accuracy: 0.01 - ETA: 0s - loss: 21377.5557 - accuracy: 0.00 - ETA: 0s - lo
ss: 22713.5814 - accuracy: 0.00 - 0s 184us/step - loss: 21743.5960 - accuracy: 0.0081
Epoch 13/100
1600/1600 [==============================] - ETA: 0s - loss: 11323.8379 - accuracy: 0.20 - ETA: 0s
- loss: 34010.2912 - accuracy: 0.01 - ETA: 0s - loss: 26084.7010 - accuracy: 0.01 - ETA: 0s -
loss: 21256.9156 - accuracy: 0.01 - ETA: 0s - loss: 21117.2134 - accuracy: 0.01 - ETA: 0s - loss:
21078.1587 - accuracy: 0.01 - 0s 181us/step - loss: 20971.2065 - accuracy: 0.0144
Epoch 14/100
1600/1600 [==============================] - ETA: 0s - loss: 16828.2500 - accuracy: 0.0000e+ - ETA
: 0s - loss: 22751.6798 - accuracy: 0.0140   - ETA: 0s - loss: 20451.4306 - accuracy: 0.00 - ETA:
0s - loss: 17532.5972 - accuracy: 0.00 - ETA: 0s - loss: 18867.6809 - accuracy: 0.00 - ETA: 0s - l
oss: 21187.5421 - accuracy: 0.00 - 0s 184us/step - loss: 20313.7531 - accuracy: 0.0094
Epoch 15/100
1600/1600 [==============================] - ETA: 0s - loss: 216753.8438 - accuracy: 0.0000e+0 - E
TA: 0s - loss: 24932.4211 - accuracy: 0.0148    - ETA: 0s - loss: 29445.2677 - accuracy: 0.01 -
ETA: 0s - loss: 26346.5855 - accuracy: 0.01 - ETA: 0s - loss: 24429.5136 - accuracy: 0.01 - ETA: 0
s - loss: 21498.7399 - accuracy: 0.02 - 0s 186us/step - loss: 19755.8592 - accuracy: 0.0244
Epoch 16/100
1600/1600 [==============================] - ETA: 0s - loss: 158278.0000 - accuracy: 0.0000e+0 - E
TA: 0s - loss: 21379.8114 - accuracy: 0.0237    - ETA: 0s - loss: 20425.2235 - accuracy: 0.04 -
ETA: 0s - loss: 20895.3035 - accuracy: 0.04 - ETA: 0s - loss: 18948.9993 - accuracy: 0.04 - ETA: 0
s - loss: 19582.5467 - accuracy: 0.04 - ETA: 0s - loss: 19570.3136 - accuracy: 0.04 - 0s
198us/step - loss: 19262.5861 - accuracy: 0.0406
Epoch 17/100
1600/1600 [==============================] - ETA: 0s - loss: 2420.1108 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 10372.4942 - accuracy: 0.0423  - ETA: 0s - loss: 15396.7039 - accuracy: 0.04 - ETA: 0
s - loss: 21646.6087 - accuracy: 0.04 - ETA: 0s - loss: 20262.7910 - accuracy: 0.05 - ETA: 0s - lo
ss: 19764.4643 - accuracy: 0.05 - 0s 189us/step - loss: 18856.2553 - accuracy: 0.0531
Epoch 18/100
1600/1600 [==============================] - ETA: 0s - loss: 851.4603 - accuracy: 0.0000e+ - ETA:
0s - loss: 15704.3097 - accuracy: 0.0691 - ETA: 0s - loss: 15171.9120 - accuracy: 0.06 - ETA: 0s -
loss: 18112.0667 - accuracy: 0.05 - ETA: 0s - loss: 16732.8908 - accuracy: 0.06 - ETA: 0s - loss:
17928.3349 - accuracy: 0.06 - 0s 179us/step - loss: 18512.2536 - accuracy: 0.0600
Epoch 19/100
1600/1600 [==============================] - ETA: 0s - loss: 4960.2095 - accuracy: 0.400 - ETA: 0s
- loss: 16501.8443 - accuracy: 0.06 - ETA: 0s - loss: 21190.3520 - accuracy: 0.06 - ETA: 0s -
loss: 23311.5806 - accuracy: 0.06 - ETA: 0s - loss: 20854.1040 - accuracy: 0.06 - ETA: 0s - loss:
18979.0953 - accuracy: 0.07 - 0s 184us/step - loss: 18218.6404 - accuracy: 0.0725
Epoch 20/100
1600/1600 [==============================] - ETA: 0s - loss: 4497.4556 - accuracy: 0.200 - ETA: 0s
- loss: 19060.2516 - accuracy: 0.09 - ETA: 0s - loss: 15004.0993 - accuracy: 0.07 - ETA: 0s -
loss: 12945.0462 - accuracy: 0.07 - ETA: 0s - loss: 14192.8917 - accuracy: 0.07 - ETA: 0s - loss:
16239.3660 - accuracy: 0.07 - 0s 186us/step - loss: 17989.5668 - accuracy: 0.0787
Epoch 21/100
1600/1600 [==============================] - ETA: 0s - loss: 37438.5859 - accuracy: 0.0000e+ - ETA
: 0s - loss: 12831.5775 - accuracy: 0.0828   - ETA: 0s - loss: 13070.1158 - accuracy: 0.07 - ETA:
0s - loss: 13489.9553 - accuracy: 0.07 - ETA: 0s - loss: 16841.5892 - accuracy: 0.08 - ETA: 0s - l
oss: 16560.2025 - accuracy: 0.08 - 0s 187us/step - loss: 17785.3183 - accuracy: 0.0838
Epoch 22/100
1600/1600 [==============================] - ETA: 0s - loss: 5658.7256 - accuracy: 0.200 - ETA: 0s
- loss: 13221.4775 - accuracy: 0.09 - ETA: 0s - loss: 15865.8015 - accuracy: 0.08 - ETA: 0s -
loss: 18145.6672 - accuracy: 0.07 - ETA: 0s - loss: 19018.0430 - accuracy: 0.08 - ETA: 0s - loss:
18355.3495 - accuracy: 0.08 - 0s 176us/step - loss: 17610.2407 - accuracy: 0.0900
Epoch 23/100
1600/1600 [==============================] - ETA: 0s - loss: 10452.4033 - accuracy: 0.0000e+ - ETA
: 0s - loss: 18086.9995 - accuracy: 0.1036   - ETA: 0s - loss: 18021.7581 - accuracy: 0.08 - ETA:
0s - loss: 17309.9359 - accuracy: 0.08 - ETA: 0s - loss: 16940.7032 - accuracy: 0.08 - ETA: 0s - l
oss: 16862.9178 - accuracy: 0.09 - 0s 183us/step - loss: 17474.3523 - accuracy: 0.0956
Epoch 24/100
1600/1600 [==============================] - ETA: 0s - loss: 12218.5547 - accuracy: 0.20 - ETA: 0s
- loss: 16783.2470 - accuracy: 0.09 - ETA: 0s - loss: 14214.9632 - accuracy: 0.10 - ETA: 0s -
loss: 15544.9070 - accuracy: 0.10 - ETA: 0s - loss: 15716.7663 - accuracy: 0.11 - ETA: 0s - loss:
16727.0475 - accuracy: 0.11 - 0s 182us/step - loss: 17353.0899 - accuracy: 0.1050
Epoch 25/100
1600/1600 [==============================] - ETA: 0s - loss: 5124.1992 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 15454.9621 - accuracy: 0.0720  - ETA: 0s - loss: 16555.1681 - accuracy: 0.08 - ETA: 0
s - loss: 20175.5514 - accuracy: 0.09 - ETA: 0s - loss: 17377.3458 - accuracy: 0.10 - ETA: 0s - lo
```

```
ss: 17685.5077 - accuracy: 0.10 - ETA: 0s - loss: 17778.8291 - accuracy: 0.10 - 0s 201us/step - lo
ss: 17253.7796 - accuracy: 0.1037
Epoch 26/100
1600/1600 [==============================] - ETA: 0s - loss: 16846.8438 - accuracy: 0.20 - ETA: 0s
- loss: 20334.9391 - accuracy: 0.12 - ETA: 0s - loss: 17715.5943 - accuracy: 0.10 - ETA: 0s -
loss: 17230.5449 - accuracy: 0.11 - ETA: 0s - loss: 16176.0493 - accuracy: 0.11 - ETA: 0s - loss:
16801.7888 - accuracy: 0.11 - ETA: 0s - loss: 17487.0754 - accuracy: 0.11 - 0s 200us/step - loss:
17161.2617 - accuracy: 0.1156
Epoch 27/100
1600/1600 [==============================] - ETA: 0s - loss: 2949.3833 - accuracy: 0.200 - ETA: 0s
- loss: 14569.8444 - accuracy: 0.12 - ETA: 0s - loss: 15157.2213 - accuracy: 0.11 - ETA: 0s -
loss: 16376.7441 - accuracy: 0.11 - ETA: 0s - loss: 16784.6538 - accuracy: 0.11 - ETA: 0s - loss:
19608.0596 - accuracy: 0.11 - ETA: 0s - loss: 17337.6686 - accuracy: 0.12 - 0s 199us/step - loss:
17088.7913 - accuracy: 0.1200
Epoch 28/100
1600/1600 [==============================] - ETA: 0s - loss: 19207.4023 - accuracy: 0.0000e+ - ETA
: 0s - loss: 13598.5574 - accuracy: 0.1167   - ETA: 0s - loss: 15053.6487 - accuracy: 0.12 - ETA:
0s - loss: 13965.5666 - accuracy: 0.11 - ETA: 0s - loss: 16698.9120 - accuracy: 0.12 - ETA: 0s - l
oss: 17794.9500 - accuracy: 0.12 - ETA: 0s - loss: 17429.4938 - accuracy: 0.12 - ETA: 0s - loss: 1
7118.3813 - accuracy: 0.12 - 0s 226us/step - loss: 17025.5701 - accuracy: 0.1269
Epoch 29/100
1600/1600 [==============================] - ETA: 0s - loss: 4317.0571 - accuracy: 0.400 - ETA: 0s
- loss: 22753.3934 - accuracy: 0.14 - ETA: 0s - loss: 17622.5406 - accuracy: 0.13 - ETA: 0s -
loss: 17296.9216 - accuracy: 0.12 - ETA: 0s - loss: 17036.8484 - accuracy: 0.12 - ETA: 0s - loss:
17666.3941 - accuracy: 0.12 - ETA: 0s - loss: 16489.1838 - accuracy: 0.12 - 0s 201us/step - loss:
16970.7756 - accuracy: 0.1275
Epoch 30/100
1600/1600 [==============================] - ETA: 0s - loss: 6323.5596 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 17415.5830 - accuracy: 0.1296  - ETA: 0s - loss: 17227.8252 - accuracy: 0.12 - ETA: 0
s - loss: 18255.1540 - accuracy: 0.12 - ETA: 0s - loss: 17690.5799 - accuracy: 0.12 - ETA: 0s - lo
ss: 17023.1743 - accuracy: 0.12 - ETA: 0s - loss: 17284.4379 - accuracy: 0.13 - 0s 200us/step - lo
ss: 16917.9300 - accuracy: 0.1312
Epoch 31/100
1600/1600 [==============================] - ETA: 0s - loss: 31170.9258 - accuracy: 0.40 - ETA: 0s
- loss: 15379.6101 - accuracy: 0.14 - ETA: 0s - loss: 17129.8082 - accuracy: 0.15 - ETA: 0s -
loss: 16661.9763 - accuracy: 0.13 - ETA: 0s - loss: 14263.9430 - accuracy: 0.13 - ETA: 0s - loss:
17250.6235 - accuracy: 0.13 - ETA: 0s - loss: 16626.8791 - accuracy: 0.13 - 0s 209us/step - loss:
16866.6008 - accuracy: 0.1344
Epoch 32/100
1600/1600 [==============================] - ETA: 0s - loss: 11816.6973 - accuracy: 0.0000e+ - ETA
: 0s - loss: 11948.9515 - accuracy: 0.1600   - ETA: 0s - loss: 15374.1090 - accuracy: 0.14 - ETA:
0s - loss: 14486.0974 - accuracy: 0.14 - ETA: 0s - loss: 14253.9533 - accuracy: 0.15 - ETA: 0s - l
oss: 16007.8313 - accuracy: 0.14 - ETA: 0s - loss: 17468.2260 - accuracy: 0.14 - ETA: 0s - loss: 1
7114.0223 - accuracy: 0.14 - 0s 232us/step - loss: 16825.6835 - accuracy: 0.1375
Epoch 33/100
1600/1600 [==============================] - ETA: 0s - loss: 22065.0566 - accuracy: 0.40 - ETA: 0s
- loss: 13232.7936 - accuracy: 0.12 - ETA: 0s - loss: 13007.0823 - accuracy: 0.14 - ETA: 0s -
loss: 18976.9312 - accuracy: 0.14 - ETA: 0s - loss: 19835.5555 - accuracy: 0.14 - ETA: 0s - loss:
17094.6793 - accuracy: 0.14 - ETA: 0s - loss: 17614.3367 - accuracy: 0.13 - 0s 218us/step - loss:
16789.7179 - accuracy: 0.1406
Epoch 34/100
1600/1600 [==============================] - ETA: 0s - loss: 1396.4137 - accuracy: 0.200 - ETA: 0s
- loss: 16306.5868 - accuracy: 0.13 - ETA: 0s - loss: 17096.5912 - accuracy: 0.13 - ETA: 0s -
loss: 17273.3725 - accuracy: 0.13 - ETA: 0s - loss: 19272.0864 - accuracy: 0.13 - ETA: 0s - loss:
19341.7791 - accuracy: 0.14 - ETA: 0s - loss: 17838.5635 - accuracy: 0.13 - ETA: 0s - loss:
17109.4153 - accuracy: 0.13 - 0s 232us/step - loss: 16743.4817 - accuracy: 0.1400
Epoch 35/100
1600/1600 [==============================] - ETA: 0s - loss: 13115.4971 - accuracy: 0.20 - ETA: 0s
- loss: 20060.3609 - accuracy: 0.10 - ETA: 0s - loss: 19009.6832 - accuracy: 0.13 - ETA: 0s -
loss: 18626.9625 - accuracy: 0.13 - ETA: 0s - loss: 20545.9047 - accuracy: 0.12 - ETA: 0s - loss:
20525.8006 - accuracy: 0.13 - ETA: 0s - loss: 18432.3235 - accuracy: 0.13 - ETA: 0s - loss:
17009.5998 - accuracy: 0.13 - 0s 238us/step - loss: 16712.0160 - accuracy: 0.1431
Epoch 36/100
1600/1600 [==============================] - ETA: 0s - loss: 5103.7393 - accuracy: 0.200 - ETA: 0s
- loss: 18807.4924 - accuracy: 0.16 - ETA: 0s - loss: 12988.0739 - accuracy: 0.15 - ETA: 0s -
loss: 12767.2305 - accuracy: 0.15 - ETA: 0s - loss: 15038.4377 - accuracy: 0.15 - ETA: 0s - loss:
14510.0434 - accuracy: 0.15 - ETA: 0s - loss: 15310.9193 - accuracy: 0.14 - ETA: 0s - loss:
16490.8311 - accuracy: 0.14 - 0s 237us/step - loss: 16681.7723 - accuracy: 0.1450
Epoch 37/100
1600/1600 [==============================] - ETA: 0s - loss: 2827.1914 - accuracy: 0.400 - ETA: 0s
- loss: 19201.7075 - accuracy: 0.15 - ETA: 0s - loss: 15895.4905 - accuracy: 0.15 - ETA: 0s -
loss: 17589.0385 - accuracy: 0.14 - ETA: 0s - loss: 19618.0285 - accuracy: 0.13 - ETA: 0s - loss:
18664.4900 - accuracy: 0.14 - ETA: 0s - loss: 16726.0299 - accuracy: 0.13 - ETA: 0s - loss:
17177.2970 - accuracy: 0.13 - 0s 239us/step - loss: 16646.8370 - accuracy: 0.1419
Epoch 38/100
1600/1600 [==============================] - ETA: 0s - loss: 1920.2551 - accuracy: 0.200 - ETA: 0s
- loss: 9702.7360 - accuracy: 0.136 - ETA: 0s - loss: 10171.8666 - accuracy: 0.16 - ETA: 0s -
```

```
loss: 15512.1498 - accuracy: 0.15 - ETA: 0s - loss: 16037.9019 - accuracy: 0.15 - ETA: 0s - loss:
17565.7934 - accuracy: 0.15 - ETA: 0s - loss: 17937.3778 - accuracy: 0.14 - ETA: 0s - loss:
16652.2045 - accuracy: 0.15 - 0s 236us/step - loss: 16614.2728 - accuracy: 0.1456
Epoch 39/100
1600/1600 [==============================] - ETA: 0s - loss: 2242.4153 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 7649.4570 - accuracy: 0.1300    - ETA: 0s - loss: 15232.5468 - accuracy: 0.13 - ETA:
0s - loss: 18143.8750 - accuracy: 0.14 - ETA: 0s - loss: 17304.1096 - accuracy: 0.13 - ETA: 0s - l
oss: 16865.7459 - accuracy: 0.14 - ETA: 0s - loss: 16755.4729 - accuracy: 0.14 - ETA: 0s - loss: 1
6722.0272 - accuracy: 0.14 - 0s 231us/step - loss: 16584.3112 - accuracy: 0.1469
Epoch 40/100
1600/1600 [==============================] - ETA: 0s - loss: 2435.0386 - accuracy: 0.200 - ETA: 0s
- loss: 6005.5799 - accuracy: 0.169 - ETA: 0s - loss: 13161.0179 - accuracy: 0.14 - ETA: 0s -
loss: 11833.4915 - accuracy: 0.14 - ETA: 0s - loss: 12866.1697 - accuracy: 0.14 - ETA: 0s - loss:
14619.9600 - accuracy: 0.14 - ETA: 0s - loss: 15135.7971 - accuracy: 0.14 - ETA: 0s - loss:
15609.4539 - accuracy: 0.14 - ETA: 0s - loss: 16576.3488 - accuracy: 0.14 - 0s 257us/step - loss:
16548.1493 - accuracy: 0.1475
Epoch 41/100
1600/1600 [==============================] - ETA: 0s - loss: 4365.9844 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 12100.9194 - accuracy: 0.1333  - ETA: 0s - loss: 20714.3752 - accuracy: 0.13 - ETA: 0
s - loss: 17503.4863 - accuracy: 0.14 - ETA: 0s - loss: 17460.5605 - accuracy: 0.14 - ETA: 0s - lo
ss: 18917.4183 - accuracy: 0.14 - ETA: 0s - loss: 18289.4428 - accuracy: 0.14 - ETA: 0s - loss: 16
766.2853 - accuracy: 0.14 - ETA: 0s - loss: 17272.4413 - accuracy: 0.14 - 0s 276us/step - loss: 16
528.7708 - accuracy: 0.1494
Epoch 42/100
1600/1600 [==============================] - ETA: 0s - loss: 4286.5044 - accuracy: 0.400 - ETA: 0s
- loss: 21859.8832 - accuracy: 0.15 - ETA: 0s - loss: 19637.7736 - accuracy: 0.16 - ETA: 0s -
loss: 18239.8235 - accuracy: 0.15 - ETA: 0s - loss: 16243.4267 - accuracy: 0.14 - ETA: 0s - loss:
16921.6034 - accuracy: 0.14 - ETA: 0s - loss: 16879.6648 - accuracy: 0.14 - ETA: 0s - loss:
15910.4374 - accuracy: 0.14 - 0s 246us/step - loss: 16494.3871 - accuracy: 0.1500
Epoch 43/100
1600/1600 [==============================] - ETA: 0s - loss: 6810.8608 - accuracy: 0.200 - ETA: 0s
- loss: 12826.3369 - accuracy: 0.15 - ETA: 0s - loss: 19239.2156 - accuracy: 0.16 - ETA: 0s -
loss: 15454.9933 - accuracy: 0.16 - ETA: 0s - loss: 16965.3166 - accuracy: 0.15 - ETA: 0s - loss:
15552.4208 - accuracy: 0.15 - ETA: 0s - loss: 14498.8138 - accuracy: 0.15 - ETA: 0s - loss:
15839.4002 - accuracy: 0.15 - ETA: 0s - loss: 16695.8815 - accuracy: 0.15 - 0s 265us/step - loss:
16474.8117 - accuracy: 0.1494
Epoch 44/100
1600/1600 [==============================] - ETA: 0s - loss: 2318.0962 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 15541.3456 - accuracy: 0.1538  - ETA: 0s - loss: 12055.6043 - accuracy: 0.15 - ETA: 0
s - loss: 18316.2236 - accuracy: 0.16 - ETA: 0s - loss: 17900.8361 - accuracy: 0.16 - ETA: 0s - lo
ss: 17379.6961 - accuracy: 0.16 - ETA: 0s - loss: 17979.2026 - accuracy: 0.16 - ETA: 0s - loss: 16
537.5465 - accuracy: 0.15 - ETA: 0s - loss: 16678.6605 - accuracy: 0.15 - ETA: 0s - loss:
16479.7680 - accuracy: 0.15 - 0s 289us/step - loss: 16447.6829 - accuracy: 0.1506
Epoch 45/100
1600/1600 [==============================] - ETA: 0s - loss: 138.2023 - accuracy: 0.20 - ETA: 0s -
loss: 13101.2929 - accuracy: 0.13 - ETA: 0s - loss: 19451.5801 - accuracy: 0.14 - ETA: 0s - loss:
16222.2759 - accuracy: 0.15 - ETA: 0s - loss: 14608.6248 - accuracy: 0.15 - ETA: 0s - loss:
13762.2516 - accuracy: 0.14 - ETA: 0s - loss: 14997.9211 - accuracy: 0.15 - ETA: 0s - loss:
14760.7489 - accuracy: 0.15 - ETA: 0s - loss: 17179.6893 - accuracy: 0.15 - 0s 278us/step - loss:
16428.0525 - accuracy: 0.1513
Epoch 46/100
1600/1600 [==============================] - ETA: 0s - loss: 8144.3203 - accuracy: 0.400 - ETA: 0s
- loss: 10964.0241 - accuracy: 0.12 - ETA: 0s - loss: 11178.9860 - accuracy: 0.14 - ETA: 0s -
loss: 13687.1274 - accuracy: 0.15 - ETA: 0s - loss: 15940.3720 - accuracy: 0.14 - ETA: 0s - loss:
14300.4656 - accuracy: 0.15 - ETA: 0s - loss: 16387.7457 - accuracy: 0.15 - ETA: 0s - loss:
16320.6282 - accuracy: 0.15 - ETA: 0s - loss: 17039.1329 - accuracy: 0.15 - 0s 271us/step - loss:
16390.3957 - accuracy: 0.1513
Epoch 47/100
1600/1600 [==============================] - ETA: 0s - loss: 8003.4204 - accuracy: 0.200 - ETA: 0s
- loss: 12450.5822 - accuracy: 0.17 - ETA: 0s - loss: 9208.5183 - accuracy: 0.1611 - ETA: 0s - los
s: 13213.4134 - accuracy: 0.15 - ETA: 0s - loss: 12544.0953 - accuracy: 0.14 - ETA: 0s - loss: 153
51.7736 - accuracy: 0.15 - ETA: 0s - loss: 16075.2276 - accuracy: 0.14 - ETA: 0s - loss:
15615.2577 - accuracy: 0.14 - ETA: 0s - loss: 17156.1170 - accuracy: 0.14 - ETA: 0s - loss:
16317.7986 - accuracy: 0.15 - 0s 290us/step - loss: 16379.3555 - accuracy: 0.1525
Epoch 48/100
1600/1600 [==============================] - ETA: 0s - loss: 7261.4502 - accuracy: 0.200 - ETA: 0s
- loss: 23541.5073 - accuracy: 0.15 - ETA: 0s - loss: 22378.3760 - accuracy: 0.15 - ETA: 0s -
loss: 19733.5849 - accuracy: 0.15 - ETA: 0s - loss: 18250.0134 - accuracy: 0.15 - ETA: 0s - loss:
17431.2327 - accuracy: 0.15 - ETA: 0s - loss: 17282.2689 - accuracy: 0.15 - ETA: 0s - loss:
16491.1252 - accuracy: 0.15 - ETA: 0s - loss: 15676.9275 - accuracy: 0.15 - 0s 274us/step - loss:
16346.3571 - accuracy: 0.1538
Epoch 49/100
1600/1600 [==============================] - ETA: 0s - loss: 2505.6912 - accuracy: 0.200 - ETA: 0s
- loss: 15216.7296 - accuracy: 0.17 - ETA: 0s - loss: 15845.4515 - accuracy: 0.18 - ETA: 0s -
loss: 13777.9547 - accuracy: 0.17 - ETA: 0s - loss: 17811.1972 - accuracy: 0.16 - ETA: 0s - loss:
15707.4555 - accuracy: 0.16 - ETA: 0s - loss: 15062.4299 - accuracy: 0.15 - ETA: 0s - loss:
16090.0810 - accuracy: 0.15 - ETA: 0s - loss: 15258.9969 - accuracy: 0.15 - ETA: 0s - loss:
```

```
16379.2173 - accuracy: 0.15 - 0s 290us/step - loss: 16331.0529 - accuracy: 0.1519
Epoch 50/100
1600/1600 [==============================] - ETA: 0s - loss: 2016.9838 - accuracy: 0.200 - ETA: 0s
- loss: 6386.7664 - accuracy: 0.139 - ETA: 0s - loss: 10114.8080 - accuracy: 0.14 - ETA: 0s -
loss: 9538.5155 - accuracy: 0.1464 - ETA: 0s - loss: 11280.9849 - accuracy: 0.16 - ETA: 0s - loss:
13339.4713 - accuracy: 0.15 - ETA: 0s - loss: 14415.6299 - accuracy: 0.15 - ETA: 0s - loss:
15672.4094 - accuracy: 0.15 - ETA: 0s - loss: 15545.5451 - accuracy: 0.15 - ETA: 0s - loss:
16058.7304 - accuracy: 0.15 - 0s 307us/step - loss: 16304.0342 - accuracy: 0.1562
Epoch 51/100
1600/1600 [==============================] - ETA: 0s - loss: 2221.5083 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 18947.1096 - accuracy: 0.1778  - ETA: 0s - loss: 13518.5686 - accuracy: 0.17 - ETA: 0
s - loss: 14929.1331 - accuracy: 0.16 - ETA: 0s - loss: 18375.0682 - accuracy: 0.16 - ETA: 0s - lo
ss: 19345.9235 - accuracy: 0.15 - ETA: 0s - loss: 19323.2497 - accuracy: 0.15 - ETA: 0s - loss: 18
576.7255 - accuracy: 0.14 - ETA: 0s - loss: 17183.1751 - accuracy: 0.15 - ETA: 0s - loss:
16328.2281 - accuracy: 0.15 - 0s 293us/step - loss: 16269.8590 - accuracy: 0.1525
Epoch 52/100
1600/1600 [==============================] - ETA: 0s - loss: 5118.4219 - accuracy: 0.200 - ETA: 0s
- loss: 6691.2979 - accuracy: 0.190 - ETA: 0s - loss: 7678.2759 - accuracy: 0.163 - ETA: 0s -
loss: 12016.7449 - accuracy: 0.15 - ETA: 0s - loss: 14796.4855 - accuracy: 0.14 - ETA: 0s - loss:
14103.6417 - accuracy: 0.15 - ETA: 0s - loss: 13163.1995 - accuracy: 0.14 - ETA: 0s - loss:
15527.1635 - accuracy: 0.15 - ETA: 0s - loss: 16568.3376 - accuracy: 0.15 - ETA: 0s - loss:
16291.6918 - accuracy: 0.15 - 0s 292us/step - loss: 16257.4168 - accuracy: 0.1556
Epoch 53/100
1600/1600 [==============================] - ETA: 0s - loss: 2329.1792 - accuracy: 0.200 - ETA: 0s
- loss: 9669.0865 - accuracy: 0.124 - ETA: 0s - loss: 12969.2877 - accuracy: 0.14 - ETA: 0s -
loss: 11722.3814 - accuracy: 0.15 - ETA: 0s - loss: 11075.8763 - accuracy: 0.15 - ETA: 0s - loss:
13396.1438 - accuracy: 0.16 - ETA: 0s - loss: 15029.4724 - accuracy: 0.15 - ETA: 0s - loss:
15931.5443 - accuracy: 0.15 - ETA: 0s - loss: 15507.7990 - accuracy: 0.15 - 0s 264us/step - loss:
16228.4095 - accuracy: 0.1562
Epoch 54/100
1600/1600 [==============================] - ETA: 0s - loss: 7111.1367 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 12389.8260 - accuracy: 0.1667  - ETA: 0s - loss: 16821.2491 - accuracy: 0.16 - ETA: 0
s - loss: 16710.9280 - accuracy: 0.15 - ETA: 0s - loss: 16114.3034 - accuracy: 0.14 - ETA: 0s - lo
ss: 14504.7755 - accuracy: 0.15 - ETA: 0s - loss: 16599.9745 - accuracy: 0.15 - ETA: 0s - loss: 15
600.8146 - accuracy: 0.15 - ETA: 0s - loss: 16403.8680 - accuracy: 0.15 - 0s 284us/step - loss: 16
201.7953 - accuracy: 0.1525
Epoch 55/100
1600/1600 [==============================] - ETA: 0s - loss: 6684.7397 - accuracy: 0.400 - ETA: 0s
- loss: 12187.0229 - accuracy: 0.19 - ETA: 0s - loss: 15119.1300 - accuracy: 0.18 - ETA: 0s -
loss: 17537.3610 - accuracy: 0.16 - ETA: 0s - loss: 20734.1046 - accuracy: 0.16 - ETA: 0s - loss:
20674.5199 - accuracy: 0.16 - ETA: 0s - loss: 18938.9006 - accuracy: 0.15 - ETA: 0s - loss:
18021.7479 - accuracy: 0.15 - ETA: 0s - loss: 16950.5079 - accuracy: 0.15 - 0s 275us/step - loss:
16193.4660 - accuracy: 0.1544
Epoch 56/100
1600/1600 [==============================] - ETA: 0s - loss: 26734.2090 - accuracy: 0.0000e+ - ETA
: 0s - loss: 7664.5861 - accuracy: 0.1030     - ETA: 0s - loss: 12181.5563 - accuracy: 0.14 - ETA:
0s - loss: 12870.1067 - accuracy: 0.14 - ETA: 0s - loss: 14105.8753 - accuracy: 0.15 - ETA: 0s - l
oss: 17008.4121 - accuracy: 0.15 - ETA: 0s - loss: 17454.7797 - accuracy: 0.15 - ETA: 0s - loss: 1
6261.8496 - accuracy: 0.15 - ETA: 0s - loss: 15716.6863 - accuracy: 0.15 - 0s 274us/step - loss: 1
6169.8959 - accuracy: 0.1556
Epoch 57/100
1600/1600 [==============================] - ETA: 0s - loss: 1904.3008 - accuracy: 0.200 - ETA: 0s
- loss: 7920.3426 - accuracy: 0.174 - ETA: 0s - loss: 8361.0626 - accuracy: 0.186 - ETA: 0s -
loss: 9822.9827 - accuracy: 0.175 - ETA: 0s - loss: 13662.1672 - accuracy: 0.17 - ETA: 0s - loss:
12522.7777 - accuracy: 0.17 - ETA: 0s - loss: 14560.3419 - accuracy: 0.16 - ETA: 0s - loss:
15142.6005 - accuracy: 0.15 - ETA: 0s - loss: 15701.2181 - accuracy: 0.15 - 0s 279us/step - loss:
16148.6042 - accuracy: 0.1550
Epoch 58/100
1600/1600 [==============================] - ETA: 0s - loss: 4603.8613 - accuracy: 0.200 - ETA: 0s
- loss: 12710.9116 - accuracy: 0.16 - ETA: 0s - loss: 9931.2773 - accuracy: 0.1784 - ETA: 0s - los
s: 11402.8716 - accuracy: 0.17 - ETA: 0s - loss: 12087.2572 - accuracy: 0.16 - ETA: 0s - loss: 160
60.7729 - accuracy: 0.15 - ETA: 0s - loss: 14560.4041 - accuracy: 0.15 - ETA: 0s - loss:
14298.7530 - accuracy: 0.15 - ETA: 0s - loss: 14028.3083 - accuracy: 0.15 - 0s 284us/step - loss:
16124.3478 - accuracy: 0.1538
Epoch 59/100
1600/1600 [==============================] - ETA: 0s - loss: 823.3109 - accuracy: 0.20 - ETA: 0s -
loss: 10403.4293 - accuracy: 0.14 - ETA: 0s - loss: 13579.7507 - accuracy: 0.15 - ETA: 0s - loss:
11736.6557 - accuracy: 0.15 - ETA: 0s - loss: 12203.6276 - accuracy: 0.16 - ETA: 0s - loss:
12331.3745 - accuracy: 0.15 - ETA: 0s - loss: 14975.3537 - accuracy: 0.16 - ETA: 0s - loss:
15362.8117 - accuracy: 0.15 - ETA: 0s - loss: 16501.7154 - accuracy: 0.15 - 0s 277us/step - loss:
16102.9197 - accuracy: 0.1550
Epoch 60/100
1600/1600 [==============================] - ETA: 0s - loss: 6972.7539 - accuracy: 0.400 - ETA: 0s
- loss: 16724.9262 - accuracy: 0.09 - ETA: 0s - loss: 17451.9368 - accuracy: 0.12 - ETA: 0s -
loss: 17496.9617 - accuracy: 0.14 - ETA: 0s - loss: 17786.6665 - accuracy: 0.14 - ETA: 0s - loss:
18137.0391 - accuracy: 0.14 - ETA: 0s - loss: 16457.0751 - accuracy: 0.14 - ETA: 0s - loss:
15835.9707 - accuracy: 0.14 - ETA: 0s - loss: 16124.9327 - accuracy: 0.15 - 0s 267us/step - loss:
```

```
16082.5891 - accuracy: 0.1544
Epoch 61/100
1600/1600 [==============================] - ETA: 0s - loss: 1494.4607 - accuracy: 0.200 - ETA: 0s
- loss: 5609.9894 - accuracy: 0.168 - ETA: 0s - loss: 9798.4102 - accuracy: 0.163 - ETA: 0s -
loss: 12905.4369 - accuracy: 0.17 - ETA: 0s - loss: 12785.9633 - accuracy: 0.16 - ETA: 0s - loss:
16747.6448 - accuracy: 0.15 - ETA: 0s - loss: 15314.4335 - accuracy: 0.15 - ETA: 0s - loss:
14705.7227 - accuracy: 0.15 - ETA: 0s - loss: 17377.0430 - accuracy: 0.16 - ETA: 0s - loss:
16876.6248 - accuracy: 0.15 - ETA: 0s - loss: 16808.9934 - accuracy: 0.15 - 1s 344us/step - loss:
16063.0401 - accuracy: 0.1550
Epoch 62/100
1600/1600 [==============================] - ETA: 0s - loss: 12147.9346 - accuracy: 0.0000e+ - ETA
: 0s - loss: 10806.3146 - accuracy: 0.1562   - ETA: 0s - loss: 13930.7098 - accuracy: 0.14 - ETA:
0s - loss: 13207.1976 - accuracy: 0.15 - ETA: 0s - loss: 16209.7380 - accuracy: 0.13 - ETA: 0s - l
oss: 15149.6361 - accuracy: 0.13 - ETA: 0s - loss: 17458.2893 - accuracy: 0.13 - ETA: 0s - loss: 1
6572.6630 - accuracy: 0.14 - ETA: 0s - loss: 17329.9398 - accuracy: 0.15 - ETA: 0s - loss:
16563.3045 - accuracy: 0.15 - 0s 301us/step - loss: 16044.6142 - accuracy: 0.1556
Epoch 63/100
1600/1600 [==============================] - ETA: 0s - loss: 3566.1353 - accuracy: 0.400 - ETA: 0s
- loss: 11428.3196 - accuracy: 0.17 - ETA: 0s - loss: 11851.0232 - accuracy: 0.15 - ETA: 0s -
loss: 11832.9426 - accuracy: 0.15 - ETA: 0s - loss: 11505.4235 - accuracy: 0.13 - ETA: 0s - loss:
12258.6761 - accuracy: 0.14 - ETA: 0s - loss: 12866.0192 - accuracy: 0.15 - ETA: 0s - loss:
13632.4532 - accuracy: 0.15 - ETA: 0s - loss: 16744.3547 - accuracy: 0.15 - ETA: 0s - loss:
15530.3534 - accuracy: 0.15 - 0s 296us/step - loss: 16023.8242 - accuracy: 0.1569
Epoch 64/100
1600/1600 [==============================] - ETA: 0s - loss: 1958.8031 - accuracy: 0.200 - ETA: 0s
- loss: 12327.4858 - accuracy: 0.11 - ETA: 0s - loss: 10490.3075 - accuracy: 0.10 - ETA: 0s -
loss: 10345.2869 - accuracy: 0.12 - ETA: 0s - loss: 13328.7737 - accuracy: 0.13 - ETA: 0s - loss:
17059.7097 - accuracy: 0.16 - ETA: 0s - loss: 18433.6802 - accuracy: 0.15 - ETA: 0s - loss:
17145.7033 - accuracy: 0.15 - ETA: 0s - loss: 16549.6581 - accuracy: 0.15 - 0s 284us/step - loss:
16014.2542 - accuracy: 0.1550
Epoch 65/100
1600/1600 [==============================] - ETA: 0s - loss: 5916.4653 - accuracy: 0.200 - ETA: 0s
- loss: 11214.1198 - accuracy: 0.19 - ETA: 0s - loss: 14785.7841 - accuracy: 0.17 - ETA: 0s -
loss: 12711.4607 - accuracy: 0.16 - ETA: 0s - loss: 11542.9943 - accuracy: 0.16 - ETA: 0s - loss:
10957.2575 - accuracy: 0.16 - ETA: 0s - loss: 12720.2639 - accuracy: 0.16 - ETA: 0s - loss:
14427.6197 - accuracy: 0.16 - ETA: 0s - loss: 15395.2642 - accuracy: 0.15 - 0s 280us/step - loss:
15990.3996 - accuracy: 0.1581
Epoch 66/100
1600/1600 [==============================] - ETA: 0s - loss: 3049.7031 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 16590.2597 - accuracy: 0.1389  - ETA: 0s - loss: 18817.9095 - accuracy: 0.14 - ETA: 0
s - loss: 18874.2307 - accuracy: 0.14 - ETA: 0s - loss: 18867.1373 - accuracy: 0.15 - ETA: 0s - lo
ss: 18433.7293 - accuracy: 0.15 - ETA: 0s - loss: 16626.8510 - accuracy: 0.16 - ETA: 0s - loss: 16
431.5260 - accuracy: 0.16 - 0s 253us/step - loss: 15967.5046 - accuracy: 0.1556
Epoch 67/100
1600/1600 [==============================] - ETA: 0s - loss: 1450.6360 - accuracy: 0.400 - ETA: 0s
- loss: 19253.3181 - accuracy: 0.15 - ETA: 0s - loss: 16266.1034 - accuracy: 0.16 - ETA: 0s -
loss: 15247.5701 - accuracy: 0.15 - ETA: 0s - loss: 14366.5877 - accuracy: 0.15 - ETA: 0s - loss:
16358.6360 - accuracy: 0.16 - ETA: 0s - loss: 16555.6502 - accuracy: 0.16 - ETA: 0s - loss:
15433.0060 - accuracy: 0.16 - ETA: 0s - loss: 15993.1157 - accuracy: 0.15 - 0s 257us/step - loss:
15952.9295 - accuracy: 0.1587
Epoch 68/100
1600/1600 [==============================] - ETA: 0s - loss: 15644.6377 - accuracy: 0.0000e+ - ETA
: 0s - loss: 11793.5283 - accuracy: 0.1053   - ETA: 0s - loss: 17790.5905 - accuracy: 0.14 - ETA:
0s - loss: 16765.7514 - accuracy: 0.15 - ETA: 0s - loss: 16506.2865 - accuracy: 0.15 - ETA: 0s - l
oss: 16214.9359 - accuracy: 0.15 - ETA: 0s - loss: 15565.1460 - accuracy: 0.15 - ETA: 0s - loss: 1
6457.1926 - accuracy: 0.14 - 0s 251us/step - loss: 15937.0274 - accuracy: 0.1569
Epoch 69/100
1600/1600 [==============================] - ETA: 0s - loss: 8576.4404 - accuracy: 0.200 - ETA: 0s
- loss: 17891.0122 - accuracy: 0.11 - ETA: 0s - loss: 12791.1299 - accuracy: 0.13 - ETA: 0s -
loss: 15343.6416 - accuracy: 0.15 - ETA: 0s - loss: 15322.0016 - accuracy: 0.15 - ETA: 0s - loss:
15597.2624 - accuracy: 0.15 - ETA: 0s - loss: 15434.1081 - accuracy: 0.15 - ETA: 0s - loss:
16602.4636 - accuracy: 0.15 - 0s 238us/step - loss: 15913.8232 - accuracy: 0.1612
Epoch 70/100
1600/1600 [==============================] - ETA: 0s - loss: 7316.1274 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 11020.9802 - accuracy: 0.1684  - ETA: 0s - loss: 12207.4274 - accuracy: 0.17 - ETA: 0
s - loss: 10998.5318 - accuracy: 0.18 - ETA: 0s - loss: 14223.7232 - accuracy: 0.18 - ETA: 0s - lo
ss: 14466.4052 - accuracy: 0.18 - ETA: 0s - loss: 13773.4848 - accuracy: 0.17 - ETA: 0s - loss: 14
076.9412 - accuracy: 0.17 - ETA: 0s - loss: 14306.7367 - accuracy: 0.16 - 0s 273us/step - loss: 15
906.5924 - accuracy: 0.1650
Epoch 71/100
1600/1600 [==============================] - ETA: 0s - loss: 1633.4535 - accuracy: 0.400 - ETA: 0s
- loss: 16611.6526 - accuracy: 0.15 - ETA: 0s - loss: 19779.9646 - accuracy: 0.15 - ETA: 0s -
loss: 17171.0074 - accuracy: 0.17 - ETA: 0s - loss: 14722.2111 - accuracy: 0.17 - ETA: 0s - loss:
14042.8980 - accuracy: 0.16 - ETA: 0s - loss: 14235.6253 - accuracy: 0.16 - 0s 215us/step - loss:
15884.3757 - accuracy: 0.1600
Epoch 72/100
1600/1600 [==============================] - ETA: 0s - loss: 2389.4851 - accuracy: 0.600 - ETA: 0s
```

```
- loss: 16233.7791 - accuracy: 0.12 - ETA: 0s - loss: 11173.3806 - accuracy: 0.15 - ETA: 0s -
loss: 9912.2914 - accuracy: 0.1652 - ETA: 0s - loss: 9872.6140 - accuracy: 0.166 - ETA: 0s - loss:
11031.1075 - accuracy: 0.15 - ETA: 0s - loss: 11674.6167 - accuracy: 0.16 - ETA: 0s - loss:
15544.0175 - accuracy: 0.16 - 0s 253us/step - loss: 15866.6198 - accuracy: 0.1650
Epoch 73/100
1600/1600 [==============================] - ETA: 0s - loss: 1234.9863 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 13764.0028 - accuracy: 0.1722  - ETA: 0s - loss: 15422.6072 - accuracy: 0.17 - ETA: 0
s - loss: 12543.9907 - accuracy: 0.17 - ETA: 0s - loss: 13765.3635 - accuracy: 0.16 - ETA: 0s - lo
ss: 14811.0520 - accuracy: 0.17 - ETA: 0s - loss: 15884.6524 - accuracy: 0.17 - 0s 216us/step - lo
ss: 15851.5378 - accuracy: 0.1688
Epoch 74/100
1600/1600 [==============================] - ETA: 0s - loss: 3294.0449 - accuracy: 0.200 - ETA: 0s
- loss: 23753.8642 - accuracy: 0.12 - ETA: 0s - loss: 18414.6131 - accuracy: 0.16 - ETA: 0s -
loss: 17821.5767 - accuracy: 0.16 - ETA: 0s - loss: 17819.8925 - accuracy: 0.16 - ETA: 0s - loss:
16376.6948 - accuracy: 0.16 - ETA: 0s - loss: 16446.0661 - accuracy: 0.16 - 0s 204us/step - loss:
15840.1349 - accuracy: 0.1650
Epoch 75/100
1600/1600 [==============================] - ETA: 0s - loss: 3250.0298 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 13863.8954 - accuracy: 0.1686  - ETA: 0s - loss: 14342.9307 - accuracy: 0.15 - ETA: 0
s - loss: 14677.6934 - accuracy: 0.17 - ETA: 0s - loss: 15493.5844 - accuracy: 0.17 - ETA: 0s - lo
ss: 15196.9304 - accuracy: 0.16 - ETA: 0s - loss: 15354.5840 - accuracy: 0.16 - 0s 202us/step - lo
ss: 15815.4857 - accuracy: 0.1688
Epoch 76/100
1600/1600 [==============================] - ETA: 0s - loss: 17569.0273 - accuracy: 0.0000e+ - ETA
: 0s - loss: 15309.3328 - accuracy: 0.1720   - ETA: 0s - loss: 16487.3709 - accuracy: 0.17 - ETA:
0s - loss: 17935.2042 - accuracy: 0.17 - ETA: 0s - loss: 16179.4213 - accuracy: 0.17 - ETA: 0s - l
oss: 16367.1970 - accuracy: 0.17 - ETA: 0s - loss: 15799.5166 - accuracy: 0.16 - 0s 195us/step - l
oss: 15794.9137 - accuracy: 0.1675
Epoch 77/100
1600/1600 [==============================] - ETA: 0s - loss: 10403.6582 - accuracy: 0.0000e+ - ETA
: 0s - loss: 12442.2956 - accuracy: 0.1957   - ETA: 0s - loss: 12822.0568 - accuracy: 0.17 - ETA:
0s - loss: 19769.7674 - accuracy: 0.16 - ETA: 0s - loss: 17586.8734 - accuracy: 0.16 - ETA: 0s - l
oss: 17082.5862 - accuracy: 0.15 - ETA: 0s - loss: 15597.6566 - accuracy: 0.16 - 0s 194us/step - l
oss: 15782.1667 - accuracy: 0.1619
Epoch 78/100
1600/1600 [==============================] - ETA: 0s - loss: 6794.8799 - accuracy: 0.200 - ETA: 0s
- loss: 16102.9970 - accuracy: 0.15 - ETA: 0s - loss: 14414.8578 - accuracy: 0.15 - ETA: 0s -
loss: 16416.8339 - accuracy: 0.14 - ETA: 0s - loss: 15660.7883 - accuracy: 0.16 - ETA: 0s - loss:
14998.4267 - accuracy: 0.17 - ETA: 0s - loss: 15814.4139 - accuracy: 0.16 - 0s 197us/step - loss:
15760.1519 - accuracy: 0.1663
Epoch 79/100
1600/1600 [==============================] - ETA: 0s - loss: 6643.3633 - accuracy: 0.200 - ETA: 0s
- loss: 8354.9476 - accuracy: 0.152 - ETA: 0s - loss: 11169.9129 - accuracy: 0.16 - ETA: 0s -
loss: 13043.9254 - accuracy: 0.16 - ETA: 0s - loss: 13976.2685 - accuracy: 0.17 - ETA: 0s - loss:
14830.5887 - accuracy: 0.17 - ETA: 0s - loss: 15795.0255 - accuracy: 0.16 - 0s 193us/step - loss:
15753.5309 - accuracy: 0.1669
Epoch 80/100
1600/1600 [==============================] - ETA: 0s - loss: 10572.2627 - accuracy: 0.0000e+ - ETA
: 0s - loss: 17778.0357 - accuracy: 0.1600   - ETA: 0s - loss: 13108.3130 - accuracy: 0.16 - ETA:
0s - loss: 13340.3500 - accuracy: 0.16 - ETA: 0s - loss: 12853.4118 - accuracy: 0.16 - ETA: 0s - l
oss: 13652.1931 - accuracy: 0.16 - 0s 183us/step - loss: 15731.9578 - accuracy: 0.1644
Epoch 81/100
1600/1600 [==============================] - ETA: 0s - loss: 19297.8711 - accuracy: 0.20 - ETA: 0s
- loss: 17238.7574 - accuracy: 0.17 - ETA: 0s - loss: 17630.4693 - accuracy: 0.15 - ETA: 0s -
loss: 15422.3904 - accuracy: 0.15 - ETA: 0s - loss: 14334.9630 - accuracy: 0.16 - ETA: 0s - loss:
15467.4414 - accuracy: 0.17 - ETA: 0s - loss: 15750.1810 - accuracy: 0.16 - 0s 192us/step - loss:
15711.0661 - accuracy: 0.1669
Epoch 82/100
1600/1600 [==============================] - ETA: 0s - loss: 4831.5771 - accuracy: 0.600 - ETA: 0s
- loss: 20262.3165 - accuracy: 0.17 - ETA: 0s - loss: 18736.2516 - accuracy: 0.18 - ETA: 0s -
loss: 19426.6544 - accuracy: 0.17 - ETA: 0s - loss: 16730.1402 - accuracy: 0.16 - ETA: 0s - loss:
16280.0016 - accuracy: 0.16 - 0s 185us/step - loss: 15697.4298 - accuracy: 0.1663
Epoch 83/100
1600/1600 [==============================] - ETA: 0s - loss: 18497.4160 - accuracy: 0.20 - ETA: 0s
- loss: 20886.7862 - accuracy: 0.16 - ETA: 0s - loss: 18347.2874 - accuracy: 0.17 - ETA: 0s -
loss: 15845.5343 - accuracy: 0.16 - ETA: 0s - loss: 17449.8547 - accuracy: 0.16 - ETA: 0s - loss:
16648.7311 - accuracy: 0.16 - 0s 189us/step - loss: 15682.4825 - accuracy: 0.1688
Epoch 84/100
1600/1600 [==============================] - ETA: 0s - loss: 142.0783 - accuracy: 0.60 - ETA: 0s -
loss: 13572.3510 - accuracy: 0.15 - ETA: 0s - loss: 20026.1891 - accuracy: 0.14 - ETA: 0s - loss:
17185.5385 - accuracy: 0.16 - ETA: 0s - loss: 16770.0959 - accuracy: 0.17 - ETA: 0s - loss:
16541.0274 - accuracy: 0.16 - 0s 188us/step - loss: 15668.1579 - accuracy: 0.1700
Epoch 85/100
1600/1600 [==============================] - ETA: 0s - loss: 1258.4646 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 13760.9583 - accuracy: 0.1860  - ETA: 0s - loss: 15860.4981 - accuracy: 0.16 - ETA: 0
s - loss: 17469.0360 - accuracy: 0.16 - ETA: 0s - loss: 16618.0682 - accuracy: 0.17 - ETA: 0s - lo
ss: 16621.9714 - accuracy: 0.16 - 0s 176us/step - loss: 15651.6161 - accuracy: 0.1663
```

```
Epoch 86/100
1600/1600 [==============================] - ETA: 0s - loss: 2580.2830 - accuracy: 0.200 - ETA: 0s
- loss: 11220.3316 - accuracy: 0.15 - ETA: 0s - loss: 14982.7435 - accuracy: 0.14 - ETA: 0s -
loss: 14740.9408 - accuracy: 0.16 - ETA: 0s - loss: 16249.3885 - accuracy: 0.16 - ETA: 0s - loss:
16448.4301 - accuracy: 0.16 - 0s 178us/step - loss: 15646.0850 - accuracy: 0.1706
Epoch 87/100
1600/1600 [==============================] - ETA: 0s - loss: 9243.2969 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 20725.6948 - accuracy: 0.1769  - ETA: 0s - loss: 18082.1243 - accuracy: 0.17 - ETA: 0
s - loss: 15811.2329 - accuracy: 0.16 - ETA: 0s - loss: 15972.3393 - accuracy: 0.17 - ETA: 0s - lo
ss: 15522.3391 - accuracy: 0.17 - 0s 175us/step - loss: 15628.5006 - accuracy: 0.1694
Epoch 88/100
1600/1600 [==============================] - ETA: 0s - loss: 12200.3105 - accuracy: 0.40 - ETA: 0s
- loss: 11273.9631 - accuracy: 0.17 - ETA: 0s - loss: 14332.7952 - accuracy: 0.17 - ETA: 0s -
loss: 14373.6886 - accuracy: 0.16 - ETA: 0s - loss: 14516.2386 - accuracy: 0.15 - ETA: 0s - loss:
15597.5988 - accuracy: 0.16 - 0s 174us/step - loss: 15618.9378 - accuracy: 0.1688
Epoch 89/100
1600/1600 [==============================] - ETA: 0s - loss: 16373.5156 - accuracy: 0.0000e+ - ETA
: 0s - loss: 9388.7170 - accuracy: 0.1811     - ETA: 0s - loss: 14733.0195 - accuracy: 0.17 - ETA:
0s - loss: 13918.7582 - accuracy: 0.16 - ETA: 0s - loss: 14993.6771 - accuracy: 0.17 - ETA: 0s - l
oss: 16221.7595 - accuracy: 0.16 - 0s 179us/step - loss: 15605.3917 - accuracy: 0.1706
Epoch 90/100
1600/1600 [==============================] - ETA: 0s - loss: 2557.2466 - accuracy: 0.200 - ETA: 0s
- loss: 14946.4644 - accuracy: 0.17 - ETA: 0s - loss: 22120.5487 - accuracy: 0.16 - ETA: 0s -
loss: 21958.7298 - accuracy: 0.16 - ETA: 0s - loss: 18464.0380 - accuracy: 0.16 - ETA: 0s - loss:
16208.3120 - accuracy: 0.16 - 0s 174us/step - loss: 15589.5052 - accuracy: 0.1706
Epoch 91/100
1600/1600 [==============================] - ETA: 0s - loss: 24084.3633 - accuracy: 0.20 - ETA: 0s
- loss: 14808.9520 - accuracy: 0.16 - ETA: 0s - loss: 15644.5634 - accuracy: 0.16 - ETA: 0s -
loss: 15900.2648 - accuracy: 0.16 - ETA: 0s - loss: 14372.9537 - accuracy: 0.15 - ETA: 0s - loss:
14746.9164 - accuracy: 0.16 - 0s 177us/step - loss: 15575.3438 - accuracy: 0.1681
Epoch 92/100
1600/1600 [==============================] - ETA: 0s - loss: 15071.2285 - accuracy: 0.0000e+ - ETA
: 0s - loss: 14223.6197 - accuracy: 0.1418   - ETA: 0s - loss: 16652.8287 - accuracy: 0.16 - ETA:
0s - loss: 14151.0778 - accuracy: 0.17 - ETA: 0s - loss: 15779.5780 - accuracy: 0.17 - ETA: 0s - l
oss: 15393.7941 - accuracy: 0.17 - ETA: 0s - loss: 15243.3722 - accuracy: 0.16 - 0s 201us/step - l
oss: 15565.4342 - accuracy: 0.1700
Epoch 93/100
1600/1600 [==============================] - ETA: 0s - loss: 2610.6570 - accuracy: 0.200 - ETA: 0s
- loss: 15989.8265 - accuracy: 0.19 - ETA: 0s - loss: 14268.5067 - accuracy: 0.16 - ETA: 0s -
loss: 16614.3796 - accuracy: 0.16 - ETA: 0s - loss: 16905.0218 - accuracy: 0.17 - ETA: 0s - loss:
16938.1561 - accuracy: 0.17 - ETA: 0s - loss: 15870.1082 - accuracy: 0.17 - 0s 201us/step - loss:
15551.0388 - accuracy: 0.1713
Epoch 94/100
1600/1600 [==============================] - ETA: 0s - loss: 5411.8662 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 13505.8517 - accuracy: 0.1774  - ETA: 0s - loss: 15918.0150 - accuracy: 0.18 - ETA: 0
s - loss: 17388.8273 - accuracy: 0.17 - ETA: 0s - loss: 17135.9506 - accuracy: 0.17 - ETA: 0s - lo
ss: 16722.2850 - accuracy: 0.16 - 0s 182us/step - loss: 15535.3346 - accuracy: 0.1694
Epoch 95/100
1600/1600 [==============================] - ETA: 0s - loss: 148231.3438 - accuracy: 0.200 - ETA:
0s - loss: 9473.4659 - accuracy: 0.2097  - ETA: 0s - loss: 10592.3516 - accuracy: 0.20 - ETA: 0s -
loss: 12864.6473 - accuracy: 0.18 - ETA: 0s - loss: 13644.4648 - accuracy: 0.17 - ETA: 0s - loss:
14700.5253 - accuracy: 0.18 - 0s 177us/step - loss: 15526.3258 - accuracy: 0.1731
Epoch 96/100
1600/1600 [==============================] - ETA: 0s - loss: 2454.9534 - accuracy: 0.200 - ETA: 0s
- loss: 11876.2815 - accuracy: 0.19 - ETA: 0s - loss: 15226.1379 - accuracy: 0.17 - ETA: 0s -
loss: 13956.5490 - accuracy: 0.16 - ETA: 0s - loss: 14258.3576 - accuracy: 0.16 - ETA: 0s - loss:
15978.6067 - accuracy: 0.17 - 0s 172us/step - loss: 15514.1147 - accuracy: 0.1700
Epoch 97/100
1600/1600 [==============================] - ETA: 0s - loss: 53316.9492 - accuracy: 0.0000e+ - ETA
: 0s - loss: 15209.3638 - accuracy: 0.1804   - ETA: 0s - loss: 17476.6055 - accuracy: 0.16 - ETA:
0s - loss: 17676.3532 - accuracy: 0.17 - ETA: 0s - loss: 17205.4532 - accuracy: 0.17 - ETA: 0s - l
oss: 15852.9732 - accuracy: 0.17 - 0s 183us/step - loss: 15504.4985 - accuracy: 0.1713
Epoch 98/100
1600/1600 [==============================] - ETA: 0s - loss: 44011.8047 - accuracy: 0.20 - ETA: 0s
- loss: 16755.7050 - accuracy: 0.21 - ETA: 0s - loss: 17707.4229 - accuracy: 0.19 - ETA: 0s -
loss: 14487.8504 - accuracy: 0.17 - ETA: 0s - loss: 16221.8345 - accuracy: 0.16 - ETA: 0s - loss:
15261.6729 - accuracy: 0.17 - 0s 180us/step - loss: 15489.1200 - accuracy: 0.1706
Epoch 99/100
1600/1600 [==============================] - ETA: 0s - loss: 5725.6807 - accuracy: 0.0000e+0 - ETA
: 0s - loss: 14879.1760 - accuracy: 0.2039  - ETA: 0s - loss: 12623.3410 - accuracy: 0.18 - ETA: 0
s - loss: 11062.1215 - accuracy: 0.17 - ETA: 0s - loss: 11763.4161 - accuracy: 0.17 - ETA: 0s - lo
ss: 14160.9117 - accuracy: 0.17 - 0s 181us/step - loss: 15479.0364 - accuracy: 0.1737
Epoch 100/100
1600/1600 [==============================] - ETA: 0s - loss: 9269.3848 - accuracy: 0.200 - ETA: 0s
- loss: 17242.9820 - accuracy: 0.18 - ETA: 0s - loss: 14335.3756 - accuracy: 0.17 - ETA: 0s -
loss: 14687.9711 - accuracy: 0.17 - ETA: 0s - loss: 14683.4375 - accuracy: 0.16 - ETA: 0s - loss:
16491.5800 - accuracy: 0.17 - ETA: 0s - loss: 16083.4149 - accuracy: 0.17 - 0s 206us/step - loss:
```

```
15464.2831 - accuracy: 0.1725
```

In [15]:

```
grid.best_params_
```

Out[15]:

```
{'batch_size': 5, 'epochs': 100}
```

In [402]:

```
predicted_test = grid.predict(X_test)

print('RMSE:', np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))
print("R Squared: ",r2_score(y_test, predicted_test))
```

```
RMSE: 178.28853372569225
R Squared:  0.030012997296552713
```

XGBOOST

In [62]:

```
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from xgboost.sklearn import XGBRegressor
import scipy.stats as st
from sklearn.model_selection import RandomizedSearchCV
```

In [63]:

```
one_to_left = st.beta(10, 1)
from_zero_positive = st.expon(0, 50)

params = {
    "n_estimators": st.randint(3, 40),
    "max_depth": st.randint(3, 40),
    "learning_rate": st.uniform(0.05, 0.4),
    "colsample_bytree": one_to_left,
    "subsample": one_to_left,
    "gamma": st.uniform(0, 10),
    'reg_alpha': from_zero_positive,
    "min_child_weight": from_zero_positive,
}

xgbreg = XGBRegressor(nthreads=-1)
gs = RandomizedSearchCV(xgbreg, params, n_jobs=1)
gs.fit(X_train, y_train)
```

```
[22:05:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

```
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[22:05:42] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[63]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
                                          n_jobs=1, nthread=None, nthreads=-1,
                                          objective='reg:...
                                        'min_child_weight':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x151efb70b8>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_froze
object at 0x151efb7320>,
                                        'reg_alpha': <scipy.stats._distn_infrastructure.rv_frozen c
ject at 0x151efb70b8>,
                                        'subsample': <scipy.stats._distn_infrastructure.rv_frozen c
ject at 0x151efb70f0>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [64]:

```
predicted_test = gs.predict(X_test)

print('RMSE:', np.sqrt(np.sum(((y_test-predicted_test)**2)/len(y_test))))
print("R Squared: ",r2_score(y_test, predicted_test))
```

```
RMSE: 149.842099563877
```

```
R Squared:  0.31484822404625645
```

Conclusion

We choose XGBOOST as out best model here. However, we can see our models performed much worse in scenario 2 The reason could be that in scenario 1, our model can successfully identify those rows that will have a spending equals to 0 and assigns a very number very close to 0. In our second case, all response numbers are not 0 so that our model will have a much larger RMSE and a lower R squred.

Problem 1C Conclusion

We can see that the models in part A usually have the better performance. I believe that models in the first part can successfully predict customers who will not spend any money and then predict a value very close to 0 to them. Then since the model will have a very small error for those customers who will not spend any money, these models will have smaller RMSE and higher R squred score. However, our purpose is predicting the money spent by those customers who will actually purchase. Hence, we shouod choose models from part B as our prefered model because it will show a better result focusing on these customers

Problem 2

Prepare Data

In [3]:

```python
data=pd.read_csv('spambase.data',header=None)
```

Feature Selection or not

Select features

In [59]:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
```

In [5]:

```python
X=data.iloc[:,:-1]
y=data.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=0, test_size=0.2)
```

In [6]:

```python
labels=np.arange(0,57)
```

In [7]:

```python
#Use random forest to find the importance of features
clf = RandomForestClassifier(n_estimators=10000, random_state=0, n_jobs=-1)

clf.fit(X_train, y_train)


for feature in zip(labels,clf.feature_importances_):
    print(feature)
```

```
(0, 0.004022923624444457)
(1, 0.0052998138046666905)
(2, 0.010383776522807557)
```

```
(2, 0.0105057705226007557)
(3, 0.0008174800236259674)
(4, 0.029597885495014162)
(5, 0.007872201694943442)
(6, 0.0820117353076857l)
(7, 0.011328159437049401)
(8, 0.004328993255793793)
(9, 0.008291416984775571)
(10, 0.010479880333745646)
(11, 0.011263558334001195)
(12, 0.003763180394301706)
(13, 0.00231991440458461)
(14, 0.0014634080165617847)
(15, 0.06565789947297779)
(16, 0.012741799762385009)
(17, 0.008416376591251329)
(18, 0.028534730758512847)
(19, 0.005015654189965477)
(20, 0.06001779698633239)
(21, 0.0025064489051581634)
(22, 0.023950022360343437)
(23, 0.03541051506717788)
(24, 0.04534252241930721)
(25, 0.017617495471516083)
(26, 0.020539078756173224)
(27, 0.005190453384489929)
(28, 0.0018801260914360397)
(29, 0.0044695863354953565)
(30, 0.001894144924322722)
(31, 0.0008475347306695996)
(32, 0.002752452138130321)
(33, 0.0008053296297909037)
(34, 0.003733454771216285)
(35, 0.004114146154301628)
(36, 0.012862677797976476)
(37, 0.0005390562152591935)
(38, 0.0035054676685030333)
(39, 0.0010699387914917462)
(40, 0.00114350084332477)
(41, 0.0052593616675887245)
(42, 0.0011252758515243606)
(43, 0.0022492355331291567)
(44, 0.009748454898895052)
(45, 0.019160566123022298)
(46, 0.0002286955427763693)
(47, 0.001349288244600137)
(48, 0.00528535001676265)
(49, 0.012774410790572729)
(50, 0.0027158205788174114)
(51, 0.12003228097262163)
(52, 0.09003222712921413)
(53, 0.0038469606469320908)
(54, 0.06479757968683204)
(55, 0.05521963176412164)
(56, 0.04237232270107911)
```

In [8]:

```
len(clf.feature_importances_[clf.feature_importances_>0.01])
#Select features with importance greater than 0.01. 23 out of
#56 features
```

Out[8]:

```
23
```

In [9]:

```
x=list(zip(labels,clf.feature_importances_))
```

In [10]:

```
def get(x):
    lst=[]
```

```
    for i in x:
        if i[1]>=0.01:
            lst.append(i[0])
    return lst
```

In [11]:

```
a=get(x)
```

In [12]:

```
X=data.iloc[:,a]
y=data.iloc[:,-1]
```

In [13]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=0, test_size=0.2)
```

Use nested to compare performance for decesion tree

In [60]:

```
from sklearn import tree
```

In [14]:

```
parameters={'max_depth': range(1,20,2)}
clf = tree.DecisionTreeClassifier(criterion="entropy", max_depth=19)
gs=GridSearchCV(clf, parameters,cv = 10, scoring = 'accuracy',n_jobs=-1)
scores=cross_val_score(gs,X,y,scoring='accuracy',cv=5)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

```
CV accuracy: 0.889 +/- 0.052
```

In [15]:

```
X=data.iloc[:,0:-2]
y=data.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=0, test_size=0.2)
```

In [16]:

```
gs=GridSearchCV(clf, parameters,cv = 10, scoring = 'accuracy',n_jobs=-1)
scores=cross_val_score(gs,X,y,scoring='accuracy',cv=5)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

```
CV accuracy: 0.899 +/- 0.044
```

Conclusion

We can see that the performance drops from 0.897 to 0.893 with 23 features. It is not a big drop but we get rid of 23 features. Hence, we choose to use feature selection.

Normalize or not using KNN to compare

In [135]:

```
X=data.iloc[:,a]
y=data.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=0, test_size=0.2)
```

In [136]:

In [136]:

```
X.head()
```

Out[136]:

	0	1	2	3	4	5	6	7	8	9	...	47	48	49	50	51	52	53	54	55	56
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.0	0.00	0.000	0.0	0.778	0.000	0.000	3.756	61	278
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.0	0.00	0.132	0.0	0.372	0.180	0.048	5.114	101	1028
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.0	0.01	0.143	0.0	0.276	0.184	0.010	9.821	485	2259
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.00	0.137	0.0	0.137	0.000	0.000	3.537	40	191
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.00	0.135	0.0	0.135	0.000	0.000	3.537	40	191

5 rows × 57 columns

In [102]:

```
k_range = list(range(1,31))
weight_options = ["uniform", "distance"]

param_grid = dict(n_neighbors = k_range, weights = weight_options)
knn = neighbors.KNeighborsClassifier()
```

In [104]:

```
grid = GridSearchCV(knn, param_grid, cv = 5, scoring = 'accuracy')
scores=cross_val_score(grid,X,y,scoring='accuracy',cv=3)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

```
CV accuracy: 0.777 +/- 0.042
```

In [105]:

```
scaler.fit(X)
X=scaler.transform(X)
```

```
C:\Games\anaconoda\lib\site-packages\sklearn\preprocessing\data.py:645: DataConversionWarning: Dat
a with input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
C:\Games\anaconoda\lib\site-packages\ipykernel_launcher.py:2: DataConversionWarning: Data with inp
ut dtype int64, float64 were all converted to float64 by StandardScaler.
```

In [106]:

```
scores=cross_val_score(grid,X,y,scoring='accuracy',cv=3)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

```
CV accuracy: 0.884 +/- 0.038
```

WE can see that, after we normalizing the data, the accuracy score increases from 0.777 to 0.884. Hence, we should normalize our dataset for models rely on distance such as KNN.

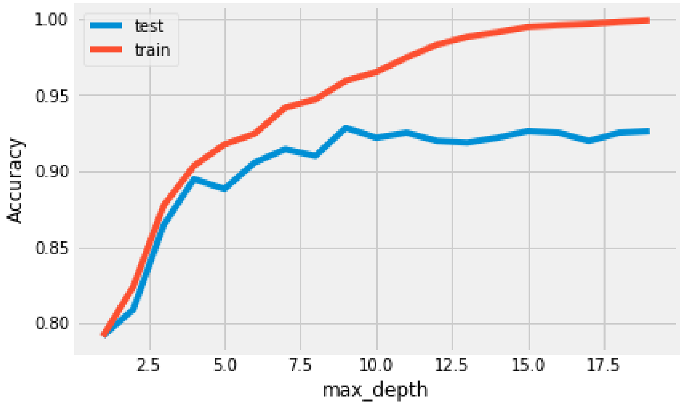
Decision Tree

In [61]:

```
X=data.iloc[:,a]
y=data.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=0, test_size=0.2)
```

In [62]:

```
complexity_values = range(1,20)
train_accuracies = []
test_accuracies=[]
# Set up possible values of parameters to optimize over
for complexity_value in complexity_values:
    clf = tree.DecisionTreeClassifier(criterion="entropy", max_depth=complexity_value)
    test_accuracies.append(clf.fit(X_train, y_train).score(X_test, y_test))
    train_accuracies.append(clf.fit(X_train, y_train).score(X_train, y_train))

line1, =plt.plot(complexity_values, test_accuracies,label='test_accuracies')
line2, =plt.plot(complexity_values, train_accuracies,label='train_accuracies')
plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.legend((line1, line2), ('test', 'train'))
plt.show()
```



In [63]:

```
clf = tree.DecisionTreeClassifier(criterion="entropy", max_depth=complexity_value)
parameters={'max_depth': range(1,20,2)}
grid = GridSearchCV(clf, parameters,cv = 10, scoring = 'accuracy')
grid.fit(X_train,y_train)
```

Out[63]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='entropy', max_depth=19,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort=False, random_state=None,
                                              splitter='best'),
             iid='warn', n_jobs=None, param_grid={'max_depth': range(1, 20, 2)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [64]:

```
print(grid.best_estimator_)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=13,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [65]:

```
clf = tree.DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=19,
            max_features=None, max_leaf_nodes=None,
```

```
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
clf.fit(X_train,y_train)
```

Out[65]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=19,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [66]:

```
from sklearn.metrics import confusion_matrix, classification_report
```

In [67]:

```
y_pred = clf.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[513  25]
 [ 44 339]]
              precision    recall  f1-score   support

           0       0.92      0.95      0.94       538
           1       0.93      0.89      0.91       383

    accuracy                           0.93       921
   macro avg       0.93      0.92      0.92       921
weighted avg       0.93      0.93      0.92       921
```

In [68]:

```
treematrix=confusion_matrix(y_test, y_pred)
```

In [69]:

```
gs=GridSearchCV(clf, parameters,cv = 10, scoring = 'accuracy',n_jobs=-1)
scores=cross_val_score(gs,X,y,scoring='accuracy',cv=5)
```

In [266]:

```
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

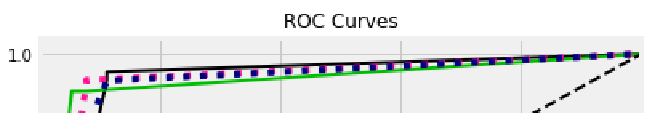
```
CV accuracy: 0.891 +/- 0.053
```

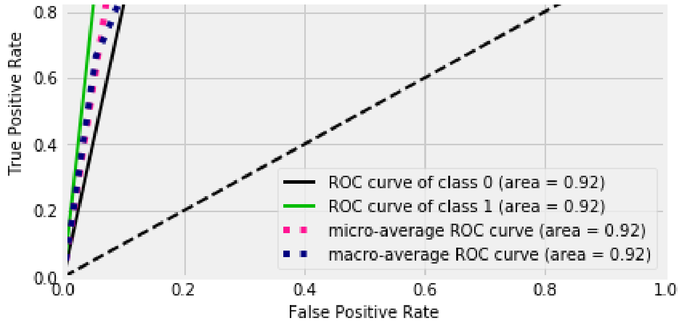
In [15]:

```
import matplotlib.pyplot as plt
import scikitplot as skplt
```

In [26]:

```
probs = clf.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, probs)
plt.show()
```





Random Forest

In [70]:

```
clf = RandomForestClassifier(n_estimators=10000, random_state=0, n_jobs=-1,max_depth=3)
```

In [71]:

```
clf.fit(X_train, y_train)
```

Out[71]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=3, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10000,
                       n_jobs=-1, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

In [72]:

```
y_pred = clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[526  12]
 [ 75 308]]
              precision    recall  f1-score   support

           0       0.88      0.98      0.92       538
           1       0.96      0.80      0.88       383

    accuracy                           0.91       921
   macro avg       0.92      0.89      0.90       921
weighted avg       0.91      0.91      0.90       921
```

In [73]:

```
rfmatrix=confusion_matrix(y_test, y_pred)
```

In [271]:

```
gs=GridSearchCV(clf, parameters,cv = 3, scoring = 'accuracy',n_jobs=-1)
scores=cross_val_score(gs,X,y,scoring='accuracy',cv=3)
```

In [272]:

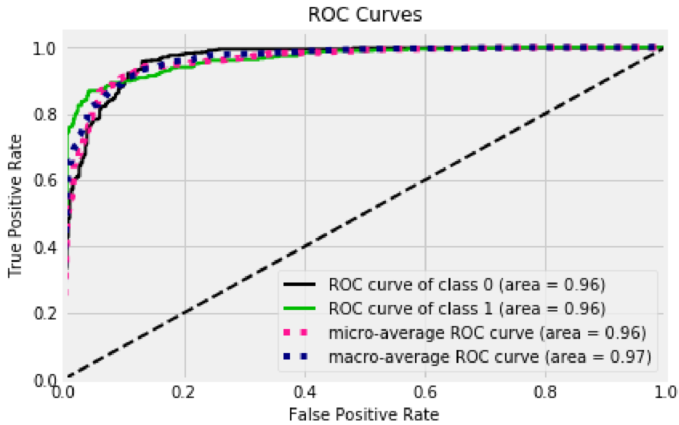
```
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

```
CV accuracy: 0.925 +/- 0.027
```

In [31]:

In [31]:

```
probs = clf.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, probs)
plt.show()
```



XGBoost

In [74]:

```
from xgboost import XGBClassifier
clf=XGBClassifier()
clf.fit(X_train,y_train)
```

Out[74]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [75]:

```
y_pred = clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[522  16]
 [ 44 339]]
              precision    recall  f1-score   support

           0       0.92      0.97      0.95       538
           1       0.95      0.89      0.92       383

    accuracy                           0.93       921
   macro avg       0.94      0.93      0.93       921
weighted avg       0.94      0.93      0.93       921
```

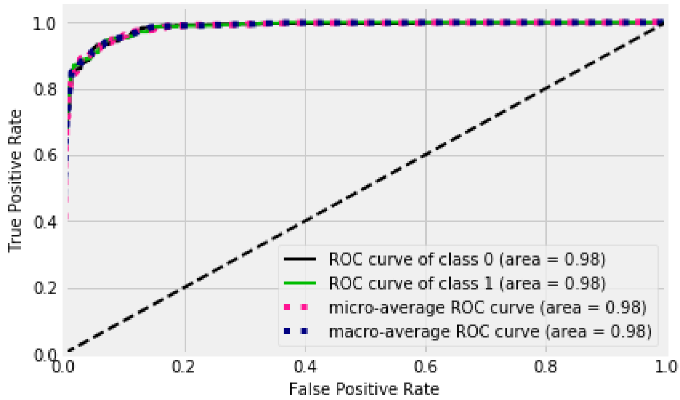
In [76]:

```
xgmatrix=confusion_matrix(y_test, y_pred)
```

In [35]:

```
probs = clf.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, probs)
plt.show()
```

ROC Curves



Neural Network

In [16]:

```
from keras.wrappers.scikit_learn import KerasClassifier
def create_model():
 # create model
 model = Sequential()
 model.add(Dense(23, input_dim=23, activation='relu'))
 model.add(Dense(1, activation='sigmoid'))

 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
 return model
model = KerasClassifier(build_fn=create_model)
```

Tune batch size and epochs

In [39]:

```
batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train, y_train)
```

```
Epoch 1/100
3680/3680 [==============================] - ETA: 29s - loss: 10.0478 - accuracy: 0.75 - ETA: 0s -
loss: 7.5182 - accuracy: 0.4878 - ETA: 0s - loss: 5.6463 - accuracy: 0.51 - ETA: 0s - loss: 4.4017
- accuracy: 0.54 - ETA: 0s - loss: 3.4854 - accuracy: 0.58 - ETA: 0s - loss: 2.8372 - accuracy: 0.
62 - 0s 116us/step - loss: 2.7376 - accuracy: 0.6380
Epoch 2/100
3680/3680 [==============================] - ETA: 0s - loss: 0.8022 - accuracy: 0.75 - ETA: 0s - l
oss: 0.8275 - accuracy: 0.78 - ETA: 0s - loss: 0.6346 - accuracy: 0.80 - ETA: 0s - loss: 0.6223 -
accuracy: 0.80 - ETA: 0s - loss: 0.5786 - accuracy: 0.81 - 0s 64us/step - loss: 0.5595 - accuracy:
0.8234
Epoch 3/100
3680/3680 [==============================] - ETA: 0s - loss: 0.7956 - accuracy: 0.75 - ETA: 0s - l
oss: 0.5447 - accuracy: 0.83 - ETA: 0s - loss: 0.4946 - accuracy: 0.83 - ETA: 0s - loss: 0.4447 -
accuracy: 0.85 - ETA: 0s - loss: 0.4196 - accuracy: 0.86 - 0s 64us/step - loss: 0.4802 - accuracy:
0.8579
Epoch 4/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3492 - accuracy: 0.90 - ETA: 0s - l
oss: 0.5290 - accuracy: 0.84 - ETA: 0s - loss: 0.4444 - accuracy: 0.85 - ETA: 0s - loss: 0.3990 -
accuracy: 0.86 - ETA: 0s - loss: 0.3932 - accuracy: 0.87 - 0s 61us/step - loss: 0.3857 - accuracy:
0.8750
Epoch 5/100
3680/3680 [==============================] - ETA: 0s - loss: 0.4349 - accuracy: 0.85 - ETA: 0s - l
oss: 0.2845 - accuracy: 0.90 - ETA: 0s - loss: 0.3437 - accuracy: 0.88 - ETA: 0s - loss: 0.3290 -
accuracy: 0.89 - ETA: 0s - loss: 0.4080 - accuracy: 0.87 - 0s 60us/step - loss: 0.4146 - accuracy:
0.8780
Epoch 6/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2958 - accuracy: 0.90 - ETA: 0s - l
oss: 0.4072 - accuracy: 0.86 - ETA: 0s - loss: 0.3585 - accuracy: 0.88 - ETA: 0s - loss: 0.3396 -
accuracy: 0.88 - ETA: 0s - loss: 0.3398 - accuracy: 0.89 - 0s 57us/step - loss: 0.3376 - accuracy:
0.8913
```

```
Epoch 7/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1483 - accuracy: 1.00 - ETA: 0s - l
oss: 0.3373 - accuracy: 0.88 - ETA: 0s - loss: 0.4290 - accuracy: 0.87 - ETA: 0s - loss: 0.4147 -
accuracy: 0.88 - 0s 54us/step - loss: 0.4066 - accuracy: 0.8813
Epoch 8/100
3680/3680 [==============================] - ETA: 0s - loss: 0.7897 - accuracy: 0.85 - ETA: 0s - l
oss: 0.2611 - accuracy: 0.90 - ETA: 0s - loss: 0.2679 - accuracy: 0.91 - ETA: 0s - loss: 0.3049 -
accuracy: 0.90 - ETA: 0s - loss: 0.5084 - accuracy: 0.87 - 0s 57us/step - loss: 0.5100 - accuracy:
0.8774
Epoch 9/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1316 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2444 - accuracy: 0.92 - ETA: 0s - loss: 0.2794 - accuracy: 0.90 - ETA: 0s - loss: 0.2893 -
accuracy: 0.90 - ETA: 0s - loss: 0.2930 - accuracy: 0.90 - 0s 58us/step - loss: 0.2936 - accuracy:
0.9035
Epoch 10/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2271 - accuracy: 0.90 - ETA: 0s - l
oss: 0.2658 - accuracy: 0.90 - ETA: 0s - loss: 0.2593 - accuracy: 0.90 - ETA: 0s - loss: 0.2720 -
accuracy: 0.90 - ETA: 0s - loss: 0.2733 - accuracy: 0.90 - 0s 57us/step - loss: 0.2737 - accuracy:
0.9057
Epoch 11/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3475 - accuracy: 0.90 - ETA: 0s - l
oss: 0.3316 - accuracy: 0.88 - ETA: 0s - loss: 0.3418 - accuracy: 0.88 - ETA: 0s - loss: 0.3801 -
accuracy: 0.88 - ETA: 0s - loss: 0.3572 - accuracy: 0.89 - 0s 59us/step - loss: 0.3551 - accuracy:
0.8929
Epoch 12/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1309 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2921 - accuracy: 0.90 - ETA: 0s - loss: 0.2774 - accuracy: 0.89 - ETA: 0s - loss: 0.2609 -
accuracy: 0.90 - ETA: 0s - loss: 0.2708 - accuracy: 0.90 - 0s 59us/step - loss: 0.2719 - accuracy:
0.9049
Epoch 13/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3047 - accuracy: 0.90 - ETA: 0s - l
oss: 0.2941 - accuracy: 0.92 - ETA: 0s - loss: 0.3952 - accuracy: 0.90 - ETA: 0s - loss: 0.3731 -
accuracy: 0.90 - ETA: 0s - loss: 0.3514 - accuracy: 0.90 - 0s 57us/step - loss: 0.3511 - accuracy:
0.9030
Epoch 14/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2619 - accuracy: 0.90 - ETA: 0s - l
oss: 0.2452 - accuracy: 0.91 - ETA: 0s - loss: 0.2977 - accuracy: 0.90 - ETA: 0s - loss: 0.2866 -
accuracy: 0.90 - ETA: 0s - loss: 0.2939 - accuracy: 0.90 - 0s 57us/step - loss: 0.2964 - accuracy:
0.9092
Epoch 15/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3930 - accuracy: 0.90 - ETA: 0s - l
oss: 0.4941 - accuracy: 0.87 - ETA: 0s - loss: 0.3831 - accuracy: 0.89 - ETA: 0s - loss: 0.3529 -
accuracy: 0.89 - 0s 52us/step - loss: 0.3330 - accuracy: 0.9000
Epoch 16/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3449 - accuracy: 0.80 - ETA: 0s - l
oss: 0.2341 - accuracy: 0.92 - ETA: 0s - loss: 0.2932 - accuracy: 0.90 - ETA: 0s - loss: 0.2988 -
accuracy: 0.90 - 0s 51us/step - loss: 0.2919 - accuracy: 0.9060
Epoch 17/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1883 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2352 - accuracy: 0.92 - ETA: 0s - loss: 0.2365 - accuracy: 0.92 - ETA: 0s - loss: 0.2382 -
accuracy: 0.91 - 0s 54us/step - loss: 0.2323 - accuracy: 0.9193
Epoch 18/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0422 - accuracy: 1.00 - ETA: 0s - l
oss: 0.2959 - accuracy: 0.91 - ETA: 0s - loss: 0.2691 - accuracy: 0.91 - ETA: 0s - loss: 0.2562 -
accuracy: 0.91 - 0s 54us/step - loss: 0.2549 - accuracy: 0.9166
Epoch 19/100
3680/3680 [==============================] - ETA: 0s - loss: 0.5329 - accuracy: 0.75 - ETA: 0s - l
oss: 0.4493 - accuracy: 0.89 - ETA: 0s - loss: 0.4194 - accuracy: 0.90 - ETA: 0s - loss: 0.3805 -
accuracy: 0.89 - 0s 52us/step - loss: 0.3711 - accuracy: 0.8989
Epoch 20/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3559 - accuracy: 0.80 - ETA: 0s - l
oss: 0.2444 - accuracy: 0.90 - ETA: 0s - loss: 0.2657 - accuracy: 0.90 - ETA: 0s - loss: 0.2478 -
accuracy: 0.90 - 0s 53us/step - loss: 0.2522 - accuracy: 0.9073
Epoch 21/100
3680/3680 [==============================] - ETA: 0s - loss: 0.4381 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2663 - accuracy: 0.90 - ETA: 0s - loss: 0.2627 - accuracy: 0.91 - ETA: 0s - loss: 0.2581 -
accuracy: 0.91 - 0s 51us/step - loss: 0.2575 - accuracy: 0.9130
Epoch 22/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2731 - accuracy: 0.90 - ETA: 0s - l
oss: 0.2102 - accuracy: 0.93 - ETA: 0s - loss: 0.2411 - accuracy: 0.92 - ETA: 0s - loss: 0.2654 -
accuracy: 0.91 - 0s 51us/step - loss: 0.2894 - accuracy: 0.9106
Epoch 23/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1973 - accuracy: 0.95 - ETA: 0s - l
oss: 0.5039 - accuracy: 0.89 - ETA: 0s - loss: 0.4382 - accuracy: 0.89 - ETA: 0s - loss: 0.4853 -
accuracy: 0.89 - 0s 52us/step - loss: 0.4714 - accuracy: 0.8938
Epoch 24/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3292 - accuracy: 0.95 - ETA: 0s - l
```

```
oss: 0.2315 - accuracy: 0.91 - ETA: 0s - loss: 0.2402 - accuracy: 0.91 - ETA: 0s - loss: 0.2717 -
accuracy: 0.91 - 0s 51us/step - loss: 0.2891 - accuracy: 0.9122
Epoch 25/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1022 - accuracy: 0.90 - ETA: 0s - l
oss: 0.3137 - accuracy: 0.90 - ETA: 0s - loss: 0.2885 - accuracy: 0.91 - ETA: 0s - loss: 0.2787 -
accuracy: 0.91 - 0s 51us/step - loss: 0.2877 - accuracy: 0.9073
Epoch 26/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1235 - accuracy: 1.00 - ETA: 0s - l
oss: 0.2332 - accuracy: 0.92 - ETA: 0s - loss: 0.2357 - accuracy: 0.92 - ETA: 0s - loss: 0.2434 -
accuracy: 0.92 - 0s 49us/step - loss: 0.2480 - accuracy: 0.9198
Epoch 27/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1296 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2796 - accuracy: 0.90 - ETA: 0s - loss: 0.3445 - accuracy: 0.90 - ETA: 0s - loss: 0.4325 -
accuracy: 0.89 - 0s 49us/step - loss: 0.5649 - accuracy: 0.8861
Epoch 28/100
3680/3680 [==============================] - ETA: 0s - loss: 1.1129 - accuracy: 0.75 - ETA: 0s - l
oss: 0.3936 - accuracy: 0.89 - ETA: 0s - loss: 0.5333 - accuracy: 0.88 - ETA: 0s - loss: 0.4670 -
accuracy: 0.89 - 0s 48us/step - loss: 0.4608 - accuracy: 0.8943
Epoch 29/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1730 - accuracy: 0.85 - ETA: 0s - l
oss: 0.2700 - accuracy: 0.92 - ETA: 0s - loss: 0.2699 - accuracy: 0.91 - ETA: 0s - loss: 0.2561 -
accuracy: 0.92 - 0s 47us/step - loss: 0.2513 - accuracy: 0.9201
Epoch 30/100
3680/3680 [==============================] - ETA: 0s - loss: 0.5714 - accuracy: 0.80 - ETA: 0s - l
oss: 0.2066 - accuracy: 0.92 - ETA: 0s - loss: 0.2562 - accuracy: 0.91 - ETA: 0s - loss: 0.3174 -
accuracy: 0.90 - 0s 45us/step - loss: 0.3110 - accuracy: 0.9071
Epoch 31/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1168 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2398 - accuracy: 0.92 - ETA: 0s - loss: 0.2367 - accuracy: 0.91 - ETA: 0s - loss: 0.2527 -
accuracy: 0.91 - 0s 46us/step - loss: 0.2658 - accuracy: 0.9182
Epoch 32/100
3680/3680 [==============================] - ETA: 0s - loss: 0.6104 - accuracy: 0.85 - ETA: 0s - l
oss: 0.3128 - accuracy: 0.92 - ETA: 0s - loss: 0.2896 - accuracy: 0.92 - ETA: 0s - loss: 0.2698 -
accuracy: 0.91 - 0s 45us/step - loss: 0.2679 - accuracy: 0.9190
Epoch 33/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0348 - accuracy: 1.00 - ETA: 0s - l
oss: 0.7479 - accuracy: 0.88 - ETA: 0s - loss: 0.6342 - accuracy: 0.88 - ETA: 0s - loss: 0.5199 -
accuracy: 0.89 - 0s 46us/step - loss: 0.4906 - accuracy: 0.8957
Epoch 34/100
3680/3680 [==============================] - ETA: 0s - loss: 0.4773 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2752 - accuracy: 0.91 - ETA: 0s - loss: 0.3118 - accuracy: 0.90 - ETA: 0s - loss: 0.3174 -
accuracy: 0.90 - 0s 46us/step - loss: 0.3042 - accuracy: 0.9122
Epoch 35/100
3680/3680 [==============================] - ETA: 0s - loss: 0.5120 - accuracy: 0.85 - ETA: 0s - l
oss: 0.2471 - accuracy: 0.92 - ETA: 0s - loss: 0.2747 - accuracy: 0.91 - ETA: 0s - loss: 0.2685 -
accuracy: 0.92 - 0s 48us/step - loss: 0.2881 - accuracy: 0.9196
Epoch 36/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1112 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2333 - accuracy: 0.92 - ETA: 0s - loss: 0.2784 - accuracy: 0.91 - ETA: 0s - loss: 0.3829 -
accuracy: 0.90 - 0s 45us/step - loss: 0.3922 - accuracy: 0.9014
Epoch 37/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2842 - accuracy: 0.90 - ETA: 0s - l
oss: 0.2134 - accuracy: 0.92 - ETA: 0s - loss: 0.2671 - accuracy: 0.91 - ETA: 0s - loss: 0.2647 -
accuracy: 0.92 - 0s 46us/step - loss: 0.2631 - accuracy: 0.9204
Epoch 38/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0653 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2087 - accuracy: 0.93 - ETA: 0s - loss: 0.2173 - accuracy: 0.93 - ETA: 0s - loss: 0.2233 -
accuracy: 0.93 - 0s 44us/step - loss: 0.2212 - accuracy: 0.9304
Epoch 39/100
3680/3680 [==============================] - ETA: 0s - loss: 0.4107 - accuracy: 0.80 - ETA: 0s - l
oss: 0.2042 - accuracy: 0.92 - ETA: 0s - loss: 0.2331 - accuracy: 0.93 - ETA: 0s - loss: 0.2460 -
accuracy: 0.92 - 0s 45us/step - loss: 0.2418 - accuracy: 0.9277
Epoch 40/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1780 - accuracy: 0.90 - ETA: 0s - l
oss: 0.2264 - accuracy: 0.93 - ETA: 0s - loss: 0.2763 - accuracy: 0.92 - ETA: 0s - loss: 0.2673 -
accuracy: 0.92 - 0s 44us/step - loss: 0.2681 - accuracy: 0.9215
Epoch 41/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2709 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2218 - accuracy: 0.93 - ETA: 0s - loss: 0.2193 - accuracy: 0.93 - ETA: 0s - loss: 0.2546 -
accuracy: 0.92 - 0s 46us/step - loss: 0.2676 - accuracy: 0.9204
Epoch 42/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2047 - accuracy: 0.95 - ETA: 0s - l
oss: 0.3300 - accuracy: 0.90 - ETA: 0s - loss: 0.2579 - accuracy: 0.91 - ETA: 0s - loss: 0.3393 -
accuracy: 0.90 - 0s 46us/step - loss: 0.3384 - accuracy: 0.9076
Epoch 43/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0518 - accuracy: 1.00 - ETA: 0s - l
oss: 0.3125 - accuracy: 0.93 - ETA: 0s - loss: 0.3373 - accuracy: 0.92 - ETA: 0s - loss: 0.3824 -
```

```
accuracy: 0.90 - 0s 44us/step - loss: 0.3859 - accuracy: 0.9090
Epoch 44/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2196 - accuracy: 0.95 - ETA: 0s - l
oss: 0.3026 - accuracy: 0.90 - ETA: 0s - loss: 0.3202 - accuracy: 0.91 - ETA: 0s - loss: 0.3127 -
accuracy: 0.91 - 0s 44us/step - loss: 0.3100 - accuracy: 0.9155
Epoch 45/100
3680/3680 [==============================] - ETA: 0s - loss: 0.4539 - accuracy: 0.80 - ETA: 0s - l
oss: 0.3018 - accuracy: 0.91 - ETA: 0s - loss: 0.3051 - accuracy: 0.91 - ETA: 0s - loss: 0.2753 -
accuracy: 0.91 - 0s 44us/step - loss: 0.2744 - accuracy: 0.9182
Epoch 46/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3070 - accuracy: 0.80 - ETA: 0s - l
oss: 0.3078 - accuracy: 0.92 - ETA: 0s - loss: 0.2585 - accuracy: 0.92 - ETA: 0s - loss: 0.2441 -
accuracy: 0.92 - 0s 44us/step - loss: 0.2396 - accuracy: 0.9283
Epoch 47/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3528 - accuracy: 0.85 - ETA: 0s - l
oss: 0.2050 - accuracy: 0.93 - ETA: 0s - loss: 0.2081 - accuracy: 0.92 - ETA: 0s - loss: 0.3377 -
accuracy: 0.91 - 0s 46us/step - loss: 0.3481 - accuracy: 0.9158
Epoch 48/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3420 - accuracy: 0.85 - ETA: 0s - l
oss: 0.4388 - accuracy: 0.89 - ETA: 0s - loss: 0.3489 - accuracy: 0.90 - ETA: 0s - loss: 0.4406 -
accuracy: 0.89 - 0s 45us/step - loss: 0.4275 - accuracy: 0.9005
Epoch 49/100
3680/3680 [==============================] - ETA: 0s - loss: 0.7933 - accuracy: 0.85 - ETA: 0s - l
oss: 0.2493 - accuracy: 0.91 - ETA: 0s - loss: 0.3501 - accuracy: 0.90 - ETA: 0s - loss: 0.3396 -
accuracy: 0.90 - 0s 45us/step - loss: 0.3418 - accuracy: 0.9106
Epoch 50/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1547 - accuracy: 0.95 - ETA: 0s - l
oss: 0.1920 - accuracy: 0.92 - ETA: 0s - loss: 0.2104 - accuracy: 0.92 - ETA: 0s - loss: 0.2229 -
accuracy: 0.92 - 0s 45us/step - loss: 0.2217 - accuracy: 0.9274
Epoch 51/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2267 - accuracy: 0.85 - ETA: 0s - l
oss: 0.2228 - accuracy: 0.92 - ETA: 0s - loss: 0.2118 - accuracy: 0.93 - ETA: 0s - loss: 0.2217 -
accuracy: 0.92 - 0s 44us/step - loss: 0.2236 - accuracy: 0.9288
Epoch 52/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1157 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2341 - accuracy: 0.91 - ETA: 0s - loss: 0.2774 - accuracy: 0.91 - ETA: 0s - loss: 0.3087 -
accuracy: 0.91 - 0s 44us/step - loss: 0.3066 - accuracy: 0.9179
Epoch 53/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0842 - accuracy: 1.00 - ETA: 0s - l
oss: 0.2383 - accuracy: 0.92 - ETA: 0s - loss: 0.2833 - accuracy: 0.92 - ETA: 0s - loss: 0.3184 -
accuracy: 0.91 - 0s 45us/step - loss: 0.3032 - accuracy: 0.9209
Epoch 54/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2358 - accuracy: 0.90 - ETA: 0s - l
oss: 0.1836 - accuracy: 0.93 - ETA: 0s - loss: 0.2222 - accuracy: 0.93 - ETA: 0s - loss: 0.2153 -
accuracy: 0.93 - 0s 43us/step - loss: 0.2171 - accuracy: 0.9340
Epoch 55/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0980 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2383 - accuracy: 0.92 - ETA: 0s - loss: 0.2325 - accuracy: 0.92 - 0s 42us/step - loss: 0.23
27 - accuracy: 0.9266
Epoch 56/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2918 - accuracy: 0.85 - ETA: 0s - l
oss: 0.1865 - accuracy: 0.93 - ETA: 0s - loss: 0.2125 - accuracy: 0.92 - 0s 42us/step - loss: 0.26
50 - accuracy: 0.9253
Epoch 57/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0674 - accuracy: 1.00 - ETA: 0s - l
oss: 0.2625 - accuracy: 0.91 - ETA: 0s - loss: 0.2727 - accuracy: 0.92 - ETA: 0s - loss: 0.3600 -
accuracy: 0.91 - 0s 43us/step - loss: 0.3603 - accuracy: 0.9158
Epoch 58/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3550 - accuracy: 0.85 - ETA: 0s - l
oss: 0.4740 - accuracy: 0.90 - ETA: 0s - loss: 0.4021 - accuracy: 0.90 - 0s 41us/step - loss: 0.38
93 - accuracy: 0.9098
Epoch 59/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0838 - accuracy: 1.00 - ETA: 0s - l
oss: 0.4751 - accuracy: 0.89 - ETA: 0s - loss: 0.3779 - accuracy: 0.90 - ETA: 0s - loss: 0.3478 -
accuracy: 0.91 - 0s 45us/step - loss: 0.3429 - accuracy: 0.9147
Epoch 60/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2009 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2772 - accuracy: 0.91 - ETA: 0s - loss: 0.2411 - accuracy: 0.92 - ETA: 0s - loss: 0.2285 -
accuracy: 0.92 - 0s 43us/step - loss: 0.2290 - accuracy: 0.9277
Epoch 61/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1371 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2096 - accuracy: 0.93 - ETA: 0s - loss: 0.2096 - accuracy: 0.93 - ETA: 0s - loss: 0.2195 -
accuracy: 0.93 - 0s 43us/step - loss: 0.2188 - accuracy: 0.9299
Epoch 62/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0995 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2521 - accuracy: 0.91 - ETA: 0s - loss: 0.2604 - accuracy: 0.92 - ETA: 0s - loss: 0.2623 -
accuracy: 0.92 - 0s 43us/step - loss: 0.2601 - accuracy: 0.9261
```

```
Epoch 63/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1251 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2020 - accuracy: 0.93 - ETA: 0s - loss: 0.2246 - accuracy: 0.92 - ETA: 0s - loss: 0.2154 -
accuracy: 0.92 - 0s 43us/step - loss: 0.2139 - accuracy: 0.9285
Epoch 64/100
3680/3680 [==============================] - ETA: 0s - loss: 0.5961 - accuracy: 0.80 - ETA: 0s - l
oss: 0.2918 - accuracy: 0.90 - ETA: 0s - loss: 0.3620 - accuracy: 0.90 - 0s 42us/step - loss: 0.30
56 - accuracy: 0.9185
Epoch 65/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0430 - accuracy: 1.00 - ETA: 0s - l
oss: 0.1906 - accuracy: 0.93 - ETA: 0s - loss: 0.1976 - accuracy: 0.93 - 0s 41us/step - loss: 0.21
01 - accuracy: 0.9353
Epoch 66/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0911 - accuracy: 1.00 - ETA: 0s - l
oss: 0.2203 - accuracy: 0.92 - ETA: 0s - loss: 0.2535 - accuracy: 0.93 - 0s 41us/step - loss: 0.23
64 - accuracy: 0.9312
Epoch 67/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1391 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2159 - accuracy: 0.92 - ETA: 0s - loss: 0.2244 - accuracy: 0.92 - ETA: 0s - loss: 0.2275 -
accuracy: 0.92 - 0s 45us/step - loss: 0.2376 - accuracy: 0.9228
Epoch 68/100
3680/3680 [==============================] - ETA: 0s - loss: 1.3272 - accuracy: 0.80 - ETA: 0s - l
oss: 0.7436 - accuracy: 0.88 - ETA: 0s - loss: 0.5523 - accuracy: 0.89 - ETA: 0s - loss: 0.4451 -
accuracy: 0.90 - 0s 43us/step - loss: 0.4395 - accuracy: 0.9060
Epoch 69/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0660 - accuracy: 1.00 - ETA: 0s - l
oss: 0.2377 - accuracy: 0.92 - ETA: 0s - loss: 0.2243 - accuracy: 0.92 - ETA: 0s - loss: 0.2273 -
accuracy: 0.92 - 0s 44us/step - loss: 0.2281 - accuracy: 0.9291
Epoch 70/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2517 - accuracy: 0.90 - ETA: 0s - l
oss: 0.1948 - accuracy: 0.93 - ETA: 0s - loss: 0.2408 - accuracy: 0.92 - ETA: 0s - loss: 0.2389 -
accuracy: 0.92 - 0s 43us/step - loss: 0.2362 - accuracy: 0.9261
Epoch 71/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1286 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2315 - accuracy: 0.94 - ETA: 0s - loss: 0.2464 - accuracy: 0.93 - ETA: 0s - loss: 0.2374 -
accuracy: 0.93 - 0s 43us/step - loss: 0.2356 - accuracy: 0.9329
Epoch 72/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1212 - accuracy: 0.95 - ETA: 0s - l
oss: 0.1762 - accuracy: 0.93 - ETA: 0s - loss: 0.1809 - accuracy: 0.94 - 0s 41us/step - loss: 0.19
75 - accuracy: 0.9375
Epoch 73/100
3680/3680 [==============================] - ETA: 0s - loss: 0.4035 - accuracy: 0.80 - ETA: 0s - l
oss: 0.1890 - accuracy: 0.93 - ETA: 0s - loss: 0.2010 - accuracy: 0.93 - 0s 41us/step - loss: 0.21
23 - accuracy: 0.9351
Epoch 74/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0863 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2247 - accuracy: 0.93 - ETA: 0s - loss: 0.2133 - accuracy: 0.93 - 0s 41us/step - loss: 0.20
19 - accuracy: 0.9353
Epoch 75/100
3680/3680 [==============================] - ETA: 0s - loss: 0.5468 - accuracy: 0.80 - ETA: 0s - l
oss: 0.2052 - accuracy: 0.93 - ETA: 0s - loss: 0.2167 - accuracy: 0.93 - ETA: 0s - loss: 0.2922 -
accuracy: 0.92 - 0s 42us/step - loss: 0.2913 - accuracy: 0.9212
Epoch 76/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1883 - accuracy: 0.90 - ETA: 0s - l
oss: 0.2057 - accuracy: 0.93 - ETA: 0s - loss: 0.1969 - accuracy: 0.93 - ETA: 0s - loss: 0.2275 -
accuracy: 0.93 - 0s 45us/step - loss: 0.2261 - accuracy: 0.9293
Epoch 77/100
3680/3680 [==============================] - ETA: 0s - loss: 0.4096 - accuracy: 0.85 - ETA: 0s - l
oss: 0.2354 - accuracy: 0.92 - ETA: 0s - loss: 0.2643 - accuracy: 0.92 - ETA: 0s - loss: 0.3276 -
accuracy: 0.91 - 0s 44us/step - loss: 0.3711 - accuracy: 0.9155
Epoch 78/100
3680/3680 [==============================] - ETA: 0s - loss: 0.4003 - accuracy: 0.90 - ETA: 0s - l
oss: 0.2750 - accuracy: 0.92 - ETA: 0s - loss: 0.2636 - accuracy: 0.93 - ETA: 0s - loss: 0.2989 -
accuracy: 0.92 - 0s 43us/step - loss: 0.2972 - accuracy: 0.9236
Epoch 79/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1911 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2591 - accuracy: 0.93 - ETA: 0s - loss: 0.2230 - accuracy: 0.93 - 0s 41us/step - loss: 0.25
86 - accuracy: 0.9321
Epoch 80/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2209 - accuracy: 0.85 - ETA: 0s - l
oss: 0.2373 - accuracy: 0.91 - ETA: 0s - loss: 0.2140 - accuracy: 0.92 - 0s 41us/step - loss: 0.20
15 - accuracy: 0.9337
Epoch 81/100
3680/3680 [==============================] - ETA: 0s - loss: 0.4258 - accuracy: 0.85 - ETA: 0s - l
oss: 0.2163 - accuracy: 0.92 - ETA: 0s - loss: 0.1966 - accuracy: 0.93 - ETA: 0s - loss: 0.2334 -
accuracy: 0.92 - 0s 42us/step - loss: 0.2419 - accuracy: 0.9277
Epoch 82/100
```

```
3680/3680 [==============================] - ETA: 0s - loss: 2.0758 - accuracy: 0.90 - ETA: 0s - l
oss: 0.4518 - accuracy: 0.91 - ETA: 0s - loss: 0.3288 - accuracy: 0.92 - ETA: 0s - loss: 0.3043 -
accuracy: 0.92 - 0s 42us/step - loss: 0.3032 - accuracy: 0.9255
Epoch 83/100
3680/3680 [==============================] - ETA: 0s - loss: 0.6339 - accuracy: 0.90 - ETA: 0s - l
oss: 0.1904 - accuracy: 0.93 - ETA: 0s - loss: 0.2065 - accuracy: 0.93 - 0s 41us/step - loss: 0.23
23 - accuracy: 0.9329
Epoch 84/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2277 - accuracy: 0.95 - ETA: 0s - l
oss: 0.1945 - accuracy: 0.93 - ETA: 0s - loss: 0.2104 - accuracy: 0.93 - ETA: 0s - loss: 0.2161 -
accuracy: 0.93 - 0s 43us/step - loss: 0.2174 - accuracy: 0.9323
Epoch 85/100
3680/3680 [==============================] - ETA: 0s - loss: 3.7858 - accuracy: 0.60 - ETA: 0s - l
oss: 0.4838 - accuracy: 0.91 - ETA: 0s - loss: 0.3588 - accuracy: 0.91 - ETA: 0s - loss: 0.2980 -
accuracy: 0.92 - 0s 43us/step - loss: 0.2950 - accuracy: 0.9266
Epoch 86/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1359 - accuracy: 1.00 - ETA: 0s - l
oss: 0.3108 - accuracy: 0.91 - ETA: 0s - loss: 0.3325 - accuracy: 0.92 - ETA: 0s - loss: 0.3857 -
accuracy: 0.91 - 0s 43us/step - loss: 0.3861 - accuracy: 0.9182
Epoch 87/100
3680/3680 [==============================] - ETA: 0s - loss: 0.4578 - accuracy: 0.85 - ETA: 0s - l
oss: 0.2924 - accuracy: 0.92 - ETA: 0s - loss: 0.2743 - accuracy: 0.91 - ETA: 0s - loss: 0.2443 -
accuracy: 0.92 - 0s 42us/step - loss: 0.2443 - accuracy: 0.9261
Epoch 88/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1060 - accuracy: 1.00 - ETA: 0s - l
oss: 0.3958 - accuracy: 0.91 - ETA: 0s - loss: 0.3585 - accuracy: 0.91 - 0s 41us/step - loss: 0.31
12 - accuracy: 0.9223
Epoch 89/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0817 - accuracy: 0.95 - ETA: 0s - l
oss: 0.1983 - accuracy: 0.93 - ETA: 0s - loss: 0.2096 - accuracy: 0.93 - ETA: 0s - loss: 0.2339 -
accuracy: 0.93 - 0s 44us/step - loss: 0.2443 - accuracy: 0.9288
Epoch 90/100
3680/3680 [==============================] - ETA: 0s - loss: 0.3117 - accuracy: 0.90 - ETA: 0s - l
oss: 0.2218 - accuracy: 0.92 - ETA: 0s - loss: 0.2373 - accuracy: 0.92 - ETA: 0s - loss: 0.2794 -
accuracy: 0.92 - 0s 43us/step - loss: 0.2775 - accuracy: 0.9250
Epoch 91/100
3680/3680 [==============================] - ETA: 0s - loss: 0.0494 - accuracy: 1.00 - ETA: 0s - l
oss: 0.2110 - accuracy: 0.93 - ETA: 0s - loss: 0.2056 - accuracy: 0.93 - ETA: 0s - loss: 0.2213 -
accuracy: 0.93 - 0s 43us/step - loss: 0.2215 - accuracy: 0.9342
Epoch 92/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1724 - accuracy: 0.90 - ETA: 0s - l
oss: 0.2352 - accuracy: 0.92 - ETA: 0s - loss: 0.2093 - accuracy: 0.93 - ETA: 0s - loss: 0.2039 -
accuracy: 0.93 - 0s 44us/step - loss: 0.2073 - accuracy: 0.9340
Epoch 93/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1275 - accuracy: 0.95 - ETA: 0s - l
oss: 0.1818 - accuracy: 0.94 - ETA: 0s - loss: 0.2048 - accuracy: 0.93 - ETA: 0s - loss: 0.2029 -
accuracy: 0.93 - 0s 45us/step - loss: 0.2021 - accuracy: 0.9351
Epoch 94/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2549 - accuracy: 0.90 - ETA: 0s - l
oss: 0.4146 - accuracy: 0.91 - ETA: 0s - loss: 0.3268 - accuracy: 0.92 - ETA: 0s - loss: 0.3045 -
accuracy: 0.92 - 0s 45us/step - loss: 0.2955 - accuracy: 0.9236
Epoch 95/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2056 - accuracy: 0.95 - ETA: 0s - l
oss: 0.1715 - accuracy: 0.93 - ETA: 0s - loss: 0.3121 - accuracy: 0.92 - ETA: 0s - loss: 0.4290 -
accuracy: 0.91 - 0s 46us/step - loss: 0.4166 - accuracy: 0.9130
Epoch 96/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2773 - accuracy: 0.90 - ETA: 0s - l
oss: 0.2899 - accuracy: 0.90 - ETA: 0s - loss: 0.2526 - accuracy: 0.92 - ETA: 0s - loss: 0.3085 -
accuracy: 0.92 - 0s 45us/step - loss: 0.3069 - accuracy: 0.9209
Epoch 97/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1947 - accuracy: 0.95 - ETA: 0s - l
oss: 0.3039 - accuracy: 0.93 - ETA: 0s - loss: 0.2434 - accuracy: 0.93 - ETA: 0s - loss: 0.2274 -
accuracy: 0.93 - 0s 45us/step - loss: 0.2251 - accuracy: 0.9372
Epoch 98/100
3680/3680 [==============================] - ETA: 0s - loss: 0.2688 - accuracy: 0.90 - ETA: 0s - l
oss: 0.1821 - accuracy: 0.94 - ETA: 0s - loss: 0.2471 - accuracy: 0.93 - ETA: 0s - loss: 0.2415 -
accuracy: 0.92 - 0s 44us/step - loss: 0.2399 - accuracy: 0.9296
Epoch 99/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1989 - accuracy: 0.95 - ETA: 0s - l
oss: 0.2098 - accuracy: 0.93 - ETA: 0s - loss: 0.1957 - accuracy: 0.93 - ETA: 0s - loss: 0.3386 -
accuracy: 0.92 - 0s 45us/step - loss: 0.3338 - accuracy: 0.9228
Epoch 100/100
3680/3680 [==============================] - ETA: 0s - loss: 0.1333 - accuracy: 0.95 - ETA: 0s - l
oss: 0.3238 - accuracy: 0.91 - ETA: 0s - loss: 0.2713 - accuracy: 0.92 - ETA: 0s - loss: 0.2523 -
accuracy: 0.92 - 0s 45us/step - loss: 0.2486 - accuracy: 0.9264
```

In [41]:

```
grid_result.best_params_
```

Out[41]:

```
{'batch_size': 20, 'epochs': 100}
```

In [20]:

```
def create_model():
 # create model
 model = Sequential()
 model.add(Dense(23, input_dim=23, activation='relu'))
 model.add(Dense(1, activation='sigmoid'))

 model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])
 return model
model = KerasClassifier(build_fn=create_model, verbose=0,epochs=100, batch_size=20)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[517  21]
 [ 47 336]]
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       538
           1       0.94      0.88      0.91       383

    accuracy                           0.93       921
   macro avg       0.93      0.92      0.92       921
weighted avg       0.93      0.93      0.93       921
```

In [24]:

```
nnmatrix = confusion_matrix(y_test, y_pred)
```

Tune learn rate

In [17]:

```
from keras import optimizers
def create_model(learn_rate=0.01):
# create model
    model = Sequential()
    model.add(Dense(23, input_dim=23, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    optimizer = keras.optimizers.Adam(lr=learn_rate)
    model.compile(loss='binary_crossentropy', optimizer=optimizer,metrics=['accuracy'])
    return model

model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=20, verbose=0)

learn_rate = [0.001, 0.01,0.05, 0.1]
param_grid = dict(learn_rate=learn_rate)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train, y_train)
```

In [18]:

```
grid_result.best_params_
#choose adamax as optimizer
```

Out[18]:

```
{'learn_rate': 0.01}
```

In [22]:

```
def create_model(learn_rate=0.01):
# create model
    model = Sequential()
    model.add(Dense(23, input_dim=23, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
    return model

model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=20, verbose=0)


model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

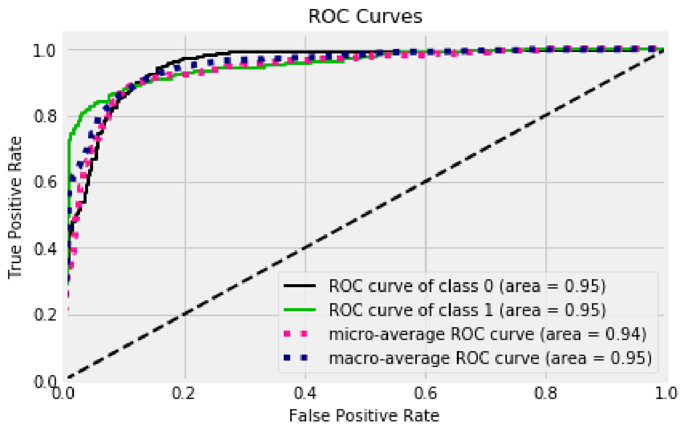
```
[[529   9]
 [ 92 291]]
              precision    recall  f1-score   support

           0       0.85      0.98      0.91       538
           1       0.97      0.76      0.85       383

    accuracy                           0.89       921
   macro avg       0.91      0.87      0.88       921
weighted avg       0.90      0.89      0.89       921
```

In [23]:

```
nnmatrix=confusion_matrix(y_test, y_pred)
```

In [24]:

```
probs = model.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, probs)
plt.show()
```



KNN

In [27]:

```
scaler.fit(X_train)
X_train_scale=scaler.transform(X_train)
```

In [28]:

```
scaler.fit(X_test)
X_test_scale=scaler.transform(X_test)
```

In [29]:

```
k_range = list(range(1,31))
weight_options = ["uniform", "distance"]

param_grid = dict(n_neighbors = k_range, weights = weight_options)
knn = neighbors.KNeighborsClassifier()

grid = GridSearchCV(knn, param_grid, cv = 10, scoring = 'accuracy')
grid.fit(X_train_scale,y_train)
y_pred =grid.predict(X_test_scale)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[513  25]
 [ 52 331]]
              precision    recall  f1-score   support

           0       0.91      0.95      0.93       538
           1       0.93      0.86      0.90       383

    accuracy                           0.92       921
   macro avg       0.92      0.91      0.91       921
weighted avg       0.92      0.92      0.92       921
```

In [30]:

```
knnmatrix=confusion_matrix(y_test, y_pred)
```

Logistic

In [38]:

```
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
```

In [ ]:

In [39]:

```
penalty = ['l1', 'l2']
C = np.logspace(0, 4, 10)
hyperparameters = dict(C=C, penalty=penalty)
```

In [40]:

```
logistic = linear_model.LogisticRegression()
clf = GridSearchCV(logistic, hyperparameters, cv=10, verbose=0)
best_model = clf.fit(X_train, y_train)
print('Best Penalty:', best_model.best_estimator_.get_params()['penalty'])
print('Best C:', best_model.best_estimator_.get_params()['C'])
```

```
Best Penalty: l1
Best C: 59.94842503189409
```

In [42]:

```
logisticRegr = LogisticRegression(C=59.94842503189409,penalty='l1')
clf = logisticRegr.fit(X_train, y_train)
clf.fit(X_train, y_train)
y_pred=clf.predict(X_test)
```

In [43]:

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

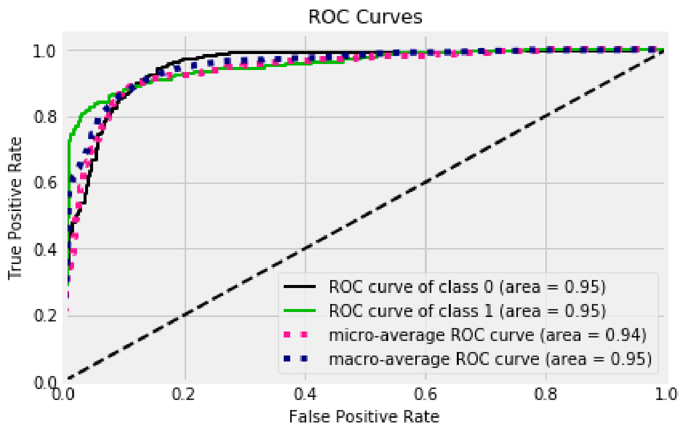
```
[[502  36]
 [ 56 327]]
              precision    recall  f1-score   support

           0       0.90      0.93      0.92       538
           1       0.90      0.85      0.88       383

    accuracy                           0.90       921
   macro avg       0.90      0.89      0.90       921
weighted avg       0.90      0.90      0.90       921
```

In [44]:

```
logitmatrix=confusion_matrix(y_test, y_pred)
```

In [45]:

```
probs = model.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, probs)
plt.show()
```



SVM

In [46]:

```
parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
'C': [1, 10, 100, 1000]},
{'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
```

In [47]:

```
from sklearn.svm import SVC
clf = GridSearchCV(SVC(), parameters, cv=5)
clf.fit(X_train_scale, y_train)
```

Out[47]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='auto_deprecated', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                          'kernel': ['rbf']},
```

```
                    {'C': [1, 10, 100, 1000], 'kernel': ['linear']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [48]:

```
print("Best parameters set found on development set:")
print()
print(clf.best_params_)
print()
print("Grid scores on development set:")
print()
```

```
Best parameters set found on development set:

{'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}

Grid scores on development set:
```

In [49]:

```
clf=SVC(kernel='rbf', C=1000,gamma=0.001)
clf.fit(X_train_scale, y_train)

y_pred = clf.predict(X_test_scale)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[515  23]
 [ 54 329]]
              precision    recall  f1-score   support

           0       0.91      0.96      0.93       538
           1       0.93      0.86      0.90       383

    accuracy                           0.92       921
   macro avg       0.92      0.91      0.91       921
weighted avg       0.92      0.92      0.92       921
```

In [52]:

```
svcmatrix=confusion_matrix(y_test, y_pred)
```

Conclusion

Based on our result, we choose XGBoost to be our best model. It has the best f1 score and best accuracy score. It also has the best AUC score based on ROC graph.

Problem 2B

Calculate Average Cost

If we mark a non-spam mail as spam mail, we might miss some important messages. Hence, we see that False Negative has higher cost

In [53]:

```
cost_matrix = np.array([[0, -10], [-1,0]])
```

In [54]:

```
cost_matrix
```

Out[54]:

```
array([[  0, -10],
       [ -1,   0]])
```

In [77]:

```
treecost= treematrix[0][1]*cost_matrix[0][1]+treematrix[1][0]*cost_matrix[1][0]
rfcost= rfmatrix[0][1]*cost_matrix[0][1]+rfmatrix[1][0]*cost_matrix[1][0]
xgcost= xgmatrix[0][1]*cost_matrix[0][1]+xgmatrix[1][0]*cost_matrix[1][0]
nncost= nnmatrix[0][1]*cost_matrix[0][1]+nnmatrix[1][0]*cost_matrix[1][0]
knncost= knnmatrix[0][1]*cost_matrix[0][1]+knnmatrix[1][0]*cost_matrix[1][0]
logitcost= logitmatrix[0][1]*cost_matrix[0][1]+logitmatrix[1][0]*cost_matrix[1][0]
svccost= svcmatrix[0][1]*cost_matrix[0][1]+svcmatrix[1][0]*cost_matrix[1][0]
```

In [78]:

```
print('tree:',treecost)
print('Random Forest:',rfcost)
print('XGBoost:',xgcost)
print('Neural Network:',nncost)
print('KNN:',knncost)
print('Logistic:',logitcost)
print('SVC:',svccost)
```

```
tree: -294
Random Forest: -195
XGBoost: -204
Neural Network: -182
KNN: -302
Logistic: -416
SVC: -284
```

Conclusion

We see that our model of Neural Network has the lease cost. We should choose Random Forest Model if we take cost into account.

Spamcase Final Write-Up

a. Data Exploration

Our data has more than 4,000 rows and 56 features. We want to predict whether a mail is a spam or not. First we see that our dataset has 56 features, which is not a lot, but still would be nice if we can manage to trim them down. Also, We see that most of our columns have numbers within two digits. However, there are several columns that with values a lot larger than other columns. We want to figure out whether we should normalize our dataset first.

b. Prepare the data

We first use a random forest model to show the importance of our data. We see that many Features have a very small importance score. We decide to use a cutoff point 0.01. We select use a nested decision tree test to compare models. With 56 features, we have an accuracy score of 0.899. With 23 features, we have an accuracy score of 0.889. We see that our accuracy score only drops by 0.01 but we filter out 50% of our features! For simplicity, we choose to use 23 features for our future analysis.

We also need to decide whether or not to use normalization. We see that the difference in scales will affect the performance of models relying on distance for calculation. Hence, we use KNN to compare performance. Before normalization, nested KNN gives us a accuracy score of 0.777. After normalization, we have an accuracy score of 0.884. Since normalization clearly has a positive impact on our model, we decide to use it for our models relying on distance for calculation.

c. Construct Models

We construct several models to compare their performance. These models include:

a. decision tree
b. Random Forest
c. XGBoost
d. Neural Network
e. KNN
f. Logistic
g. SVM.

For each model, we use cross validation to select best parameters for them. After fitting the model with the best parameters, we construct confusion matrix and ROC graph so that we have accuracy score, precision, recall, and F1 score to compare between models.

d. Compare Models

Model	**Accuracy Score**	**AUC Score**	**F1 Positive**
Decision Tree	**0.93**	**0.92**	**0.91**
Random Forest	**0.91**	**0.96**	**0.88**
XGBoost	**0.93**	**0.98**	**0.92**
Neural Network	**0.89**	**0.95**	**0.85**
KNN	**0.92**		**0.9**
Logistic	**0.9**	**0.95**	**0.88**
SVM	**0.92**		**0.9**

From the table above we see that we should choose XGBoost as our best model. It has the best score in all three categories.

e. Take Cost into Account

We have a cost of 10 and a cost of 1. We believe that if we identify a non-spam mail as spam, we will put it into spam folder and our users can miss some important information. Hence, the cost of identifying a non-spam mail as spam should be 10 and the cost of identifying a spam mail as non-spam is 1. We then have following cost result for each model.

Model	**Cost**
Decision Tree	294
Random Forest	195
XGBoost	204
Neural Network	182
KNN	302
Logistic	416
SVM	284

As we can see, the Neural Network model has the lowest cost of 182. If we want to minimize the cost, we should consider choosing it as our model. However, the cost of random forest and XGBoost models are just a little bit more than Neural Network but they can have a better performance predicting all emails. Hence, we need to consider what our goal is. If our most important and ultimate goal is minimizing the cost, we should choose Neural Network model. If we want to minimize the cost and have a model with a better overall performance, we can choose Random Forest and XGBoost as well.