In [3]:
```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import warnings
from sklearn.model_selection import GridSearchCV
warnings.filterwarnings('ignore')
from keras.layers import Dense, Dropout, Activation
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import pyodbc
from imblearn.under_sampling import RandomUnderSampler
```

In [6]:
```
host = 'getdata.cityoftulsa.org'
port = 31010
uid = '####'
pwd = '####'
driver = 'Dremio Connector'

cnxn = pyodbc.connect("Driver={};ConnectionType=Direct;HOST={};PORT={};Authen

sql = '''SELECT * from "Data Governance Stage".Engineering.UDP."UDP-6-FireDat

df = pd.read_sql(sql,cnxn)
```

In [8]:
```
import datetime
df = df[df['Data_Year']!=datetime.date(2017, 12, 31)][['SALEP','GROSSSF','Hou
```

In [10]:
```
y = df['Fire']
X = df.drop(columns=['Fire'])

X = X.fillna(0)
rus = RandomUnderSampler(random_state=10)
```

In [11]:
```
del df
```

In [12]:
```
X, y = rus.fit_resample(X, y)
```

In [13]:
```
X = pd.DataFrame(X, columns = ['SALEP','GROSSSF','HouseAge','YearsFromSale','
```

In [14]:
```
X = X.astype({'SALEP':'float64'})
X = X.astype({'GROSSSF':'float64'})
X = X.astype({'HouseAge':'int64'})
X = X.astype({'YearsFromSale':'float64'})
X = X.astype({'ACCTTYPE':'object'})
X = X.astype({'HOMESTEAD':'object'})
X = X.astype({'Tract':'object'})
```

In [15]:
```
X = pd.get_dummies(X)
```

In [16]:
```
columns = X.columns
```

In [17]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
X = scaler.fit_transform(X)
X = pd.DataFrame(X, columns = columns)
```

In [18]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
```

In [19]:
```
from sklearn.feature_selection import f_classif
result = f_classif(X,y)
```

In [20]:
```
for score, fname in sorted(zip(result[0], X.columns), reverse=True)[:10]:
    print(fname, score)
```

```
HouseAge 1057.4643317031237
HOMESTEAD_  235.0125057056941
HOMESTEAD_Homestead 235.0125057056932
Tract_40143007900 118.94928480759239
Tract_40143005807 98.24741251848201
Tract_40143000400 76.45774925362083
Tract_40143006707 72.3073809189311
Tract_40143005402 61.894209626191056
Tract_40143006200 46.35248220537134
Tract_40143006708 45.48186906789974
```

In [21]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,stratify=encoded_Y,
```

In [22]:
```
del X
del y
del encoded_Y
```

Random Forest

In [23]:
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=500, max_depth=10,oob_score=True)
```

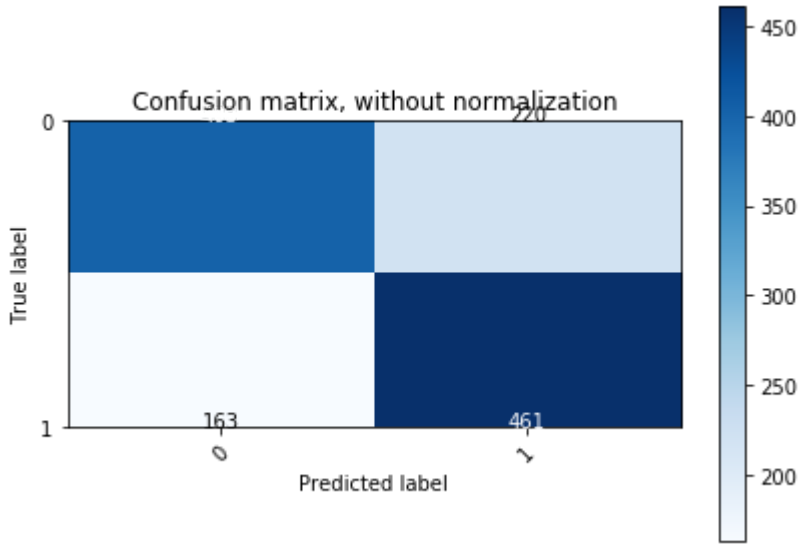
In [24]:
```
y_hat_4 = clf.fit(X_train,y_train)
```

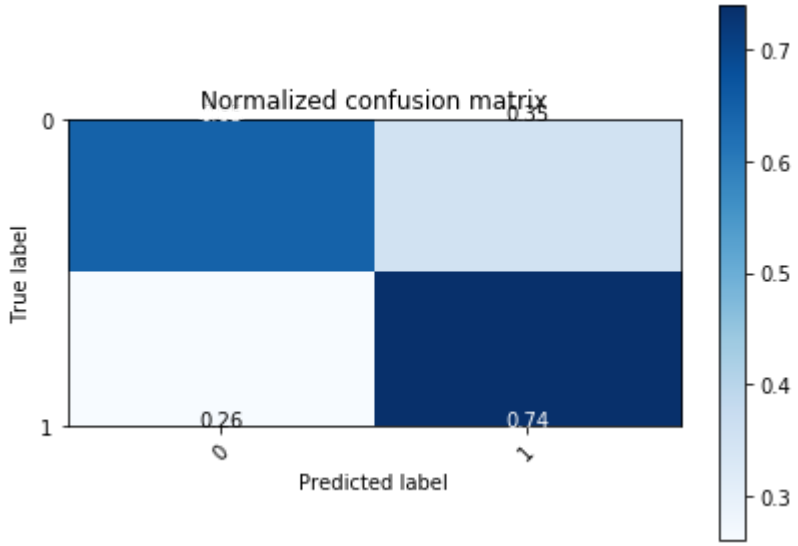
In [25]:
```
y_hat_4 = clf.predict(X_test)
```

In [26]:
```
import matplotlib.pyplot as plt
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

In [27]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
corr = confusion_matrix(y_test.reshape(y_test.shape[0]  ,1),y_hat_4.round())
np.set_printoptions(precision=2)
class_names = [0,1]
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(corr, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(corr, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[403 220]
 [163 461]]
Normalized confusion matrix
[[0.65 0.35]
 [0.26 0.74]]
```





In [29]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test.reshape(y_test.shape[0]  ,1),y_hat_4.round
```

```
              precision    recall  f1-score   support

           0       0.71      0.65      0.68       623
           1       0.68      0.74      0.71       624

    accuracy                           0.69      1247
   macro avg       0.69      0.69      0.69      1247
weighted avg       0.69      0.69      0.69      1247
```

In [40]:
```
from joblib import dump, load
dump(clf, 'Prediction/RF-FirePredModel.joblib')
```

Out[40]:
```
['Prediction/RF-FirePredModel.joblib']
```

In [30]:
```
print("Features sorted by their score:")
print(sorted(zip(map(lambda x: round(x, 4), clf.feature_importances_), X_trai
             reverse=True))
```

```
Features sorted by their score:
[(0.2836, 'HouseAge'), (0.1536, 'SALEP'), (0.1068, 'GROSSSF'), (0.05, 'HOMES
TEAD_ '), (0.0483, 'HOMESTEAD_Homestead'), (0.0232, 'Tract_40143007900'),
(0.0232, 'Tract_40143005807'), (0.0179, 'Tract_40143005402'), (0.016, 'Tract
_40143006707'), (0.0153, 'Tract_40143000400'), (0.0102, 'ACCTTYPE_Commercia
l'), (0.0093, 'Tract_40143009101'), (0.0081, 'Tract_40143008002'), (0.0074,
'Tract_40143006708'), (0.0071, 'Tract_40143006200'), (0.0068, 'Tract_4014300
5801'), (0.0066, 'ACCTTYPE_ '), (0.0063, 'Tract_40143009401'), (0.006, 'Trac
t_40143007635'), (0.006, 'Tract_40143005805'), (0.0058, 'Tract_4014300140
0'), (0.0053, 'Tract_40143007515'), (0.0051, 'ACCTTYPE_Residential'), (0.004
9, 'YearsFromSale'), (0.0049, 'Tract_40143009300'), (0.0047, 'Tract_40143009
003'), (0.0047, 'Tract_40143007702'), (0.0047, 'Tract_40143007510'), (0.004
7, 'Tract_40143006703'), (0.0046, 'Tract_40143009402'), (0.0038, 'Tract_4014
3009009'), (0.0037, 'Tract_40143007516'), (0.0037, 'Tract_40143001600'), (0.
0036, 'Tract_40143000900'), (0.0033, 'Tract_40143009007'), (0.0033, 'Tract_4
0143007701'), (0.0032, 'ACCTTYPE_Agricultural'), (0.0031, 'Tract_4014300920
0'), (0.003, 'Tract_40143003000'), (0.003, 'Tract_40143000700'), (0.0028, 'T
ract_40143007802'), (0.0026, 'Tract_40143005500'), (0.0026, 'Tract_401430002
00'), (0.0025, 'Tract_40143000300'), (0.0023, 'Tract_40143002500'), (0.0023,
'Tract_40143001500'), (0.0022, 'Tract_40143007512'), (0.0021, 'Tract_4014300
1000'), (0.0021, 'Tract_40143000100'), (0.002, 'Tract_40143009006'), (0.002,
'Tract_40143009004'), (0.002, 'Tract_40143001300'), (0.0019, 'Tract_40143007
609'), (0.0019, 'Tract_40143000500'), (0.0018, 'Tract_40143007513'), (0.001
7, 'Tract_40143009500'), (0.0017, 'Tract_40143007503'), (0.0016, 'Tract_4014
3007518'), (0.0016, 'Tract_40143005700'), (0.0015, 'Tract_40143007402'), (0.
0014, 'Tract_40143007630'), (0.0014, 'Tract_40143006907'), (0.0014, 'Tract_4
0143006705'), (0.0013, 'Tract_40143007625'), (0.0013, 'Tract_40143004200'),
(0.0012, 'Tract_40143007520'), (0.0012, 'Tract_40143007415'), (0.0012, 'Trac
t_40143007414'), (0.0012, 'Tract_40143007413'), (0.0012, 'Tract_4014300560
0'), (0.0011, 'Tract_40143008501'), (0.0011, 'Tract_40143007304'), (0.0011,
'Tract_40143005002'), (0.001, 'Tract_40143011100'), (0.001, 'Tract_401430076
19'), (0.001, 'Tract_40143006905'), (0.001, 'Tract_40143006701'), (0.001, 'T
ract_40143002301'), (0.0009, 'Tract_40143007409'), (0.0009, 'Tract_401430038
00'), (0.0009, 'ACCTTYPE_Exempt'), (0.0008, 'Tract_40143007631'), (0.0008,
'Tract_40143007629'), (0.0008, 'Tract_40143007620'), (0.0008, 'Tract_4014300
7613'), (0.0008, 'Tract_40143007519'), (0.0008, 'Tract_40143007308'), (0.000
8, 'Tract_40143004800'), (0.0008, 'Tract_40143002700'), (0.0008, 'Tract_4014
3000600'), (0.0007, 'Tract_40143007642'), (0.0007, 'Tract_40143007614'), (0.
0007, 'Tract_40143007608'), (0.0007, 'Tract_40143006801'), (0.0007, 'Tract_4
0143006600'), (0.0007, 'Tract_40143006506'), (0.0007, 'Tract_40143004301'),
(0.0006, 'Tract_40143007615'), (0.0006, 'Tract_40143007412'), (0.0006, 'Trac
t_40143007305'), (0.0006, 'Tract_40143006903'), (0.0006, 'Tract_4014300580
8'), (0.0005, 'Tract_40143008700'), (0.0005, 'Tract_40143008502'), (0.0005,
'Tract_40143008300'), (0.0005, 'Tract_40143007633'), (0.0005, 'Tract_4014300
7618'), (0.0005, 'Tract_40143007511'), (0.0005, 'Tract_40143007506'), (0.000
5, 'Tract_40143007312'), (0.0005, 'Tract_40143007311'), (0.0005, 'Tract_4014
3007200'), (0.0005, 'Tract_40143007102'), (0.0005, 'Tract_40143006803'), (0.
0005, 'Tract_40143005401'), (0.0005, 'Tract_40143004600'), (0.0005, 'Tract_4
0143003300'), (0.0005, 'Tract_40143003100'), (0.0005, 'Tract_40143001200'),
(0.0005, 'Tract_40143000800'), (0.0004, 'Tract_40143007616'), (0.0004, 'Trac
t_40143007507'), (0.0004, 'Tract_40143007407'), (0.0004, 'Tract_4014300731
0'), (0.0004, 'Tract_40143007306'), (0.0004, 'Tract_40143007101'), (0.0004,
'Tract_40143007000'), (0.0004, 'Tract_40143006804'), (0.0004, 'Tract_4014300
```

```
5200'), (0.0004, 'Tract_40143004900'), (0.0004, 'Tract_40143004101'), (0.000
4, 'Tract_40143003900'), (0.0004, 'Tract_40143003600'), (0.0004, 'Tract_4014
3003200'), (0.0003, 'Tract_40143009104'), (0.0003, 'Tract_40143009008'), (0.
0003, 'Tract_40143008900'), (0.0003, 'Tract_40143008800'), (0.0003, 'Tract_4
0143008600'), (0.0003, 'Tract_40143008200'), (0.0003, 'Tract_40143007624'),
(0.0003, 'Tract_40143007611'), (0.0003, 'Tract_40143007309'), (0.0003, 'Trac
t_40143006901'), (0.0003, 'Tract_40143006000'), (0.0003, 'Tract_4014300530
0'), (0.0003, 'Tract_40143004500'), (0.0003, 'Tract_40143004302'), (0.0003,
'Tract_40143003700'), (0.0003, 'Tract_40143003500'), (0.0003, 'Tract_4014300
1800'), (0.0003, 'ACCTTYPE_Exempt Com'), (0.0002, 'Tract_40143008400'), (0.0
002, 'Tract_40143008001'), (0.0002, 'Tract_40143007801'), (0.0002, 'Tract_40
143007639'), (0.0002, 'Tract_40143007638'), (0.0002, 'Tract_40143007637'),
(0.0002, 'Tract_40143007634'), (0.0002, 'Tract_40143007632'), (0.0002, 'Trac
t_40143007617'), (0.0002, 'Tract_40143007524'), (0.0002, 'Tract_4014300750
8'), (0.0002, 'Tract_40143006906'), (0.0002, 'Tract_40143006507'), (0.0002,
'Tract_40143005900'), (0.0002, 'Tract_40143005001'), (0.0002, 'Tract_4014300
4700'), (0.0002, 'Tract_40143004000'), (0.0002, 'Tract_40143002900'), (0.000
2, 'Tract_40143002100'), (0.0002, 'Tract_40143001900'), (0.0001, 'Tract_4014
3007641'), (0.0001, 'Tract_40143007636'), (0.0001, 'Tract_40143007612'), (0.
0001, 'Tract_40143007523'), (0.0001, 'Tract_40143007522'), (0.0001, 'Tract_4
0143007411'), (0.0001, 'Tract_40143007410'), (0.0001, 'Tract_40143007408'),
(0.0001, 'Tract_40143006902'), (0.0001, 'Tract_40143005806'), (0.0001, 'Trac
t_40143005100'), (0.0001, 'Tract_40143004400'), (0.0001, 'Tract_4014300340
0'), (0.0001, 'Tract_40143002000'), (0.0001, 'Tract_40143001700'), (0.0001,
'ACCTTYPE_Comm Res'), (0.0, 'ACCTTYPE_Partial Exempt'), (0.0, 'ACCTTYPE_Exem
pt Res'), (0.0, 'ACCTTYPE_Exempt Ag')]
```

Ridge Regression

In [31]:
```
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import RidgeClassifierCV
```

In [32]:
```
clf = RidgeClassifierCV(alphas=[1, 1e3, 1e6], store_cv_values=True).fit(X_tra
```

In [33]:
```
cv_mse = np.mean(clf.cv_values_, axis=0)
print(cv_mse)
```

```
[[0.72 0.91 1.  ]]
```

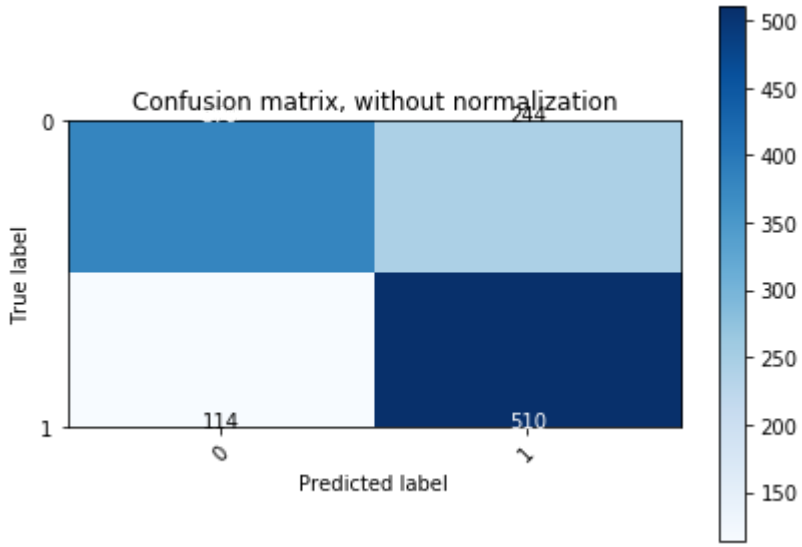
In [34]:
```
clf.score(X_test, y_test)
```

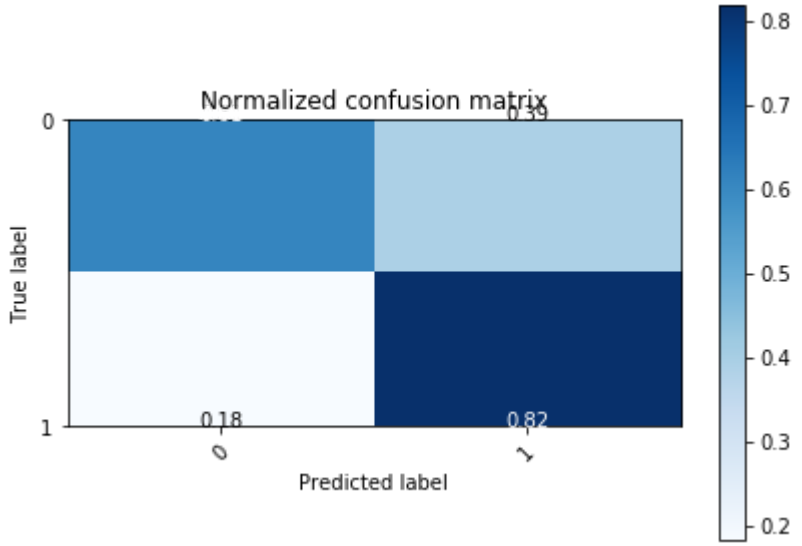
Out[34]:
```
0.7129109863672815
```

In [35]:
```
y_hat_3 = clf.predict(X_test)
```

In [36]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
corr = confusion_matrix(y_test.reshape(y_test.shape[0] ,1),y_hat_3.round())
np.set_printoptions(precision=2)
class_names = [0,1]
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(corr, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(corr, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[379 244]
 [114 510]]
Normalized confusion matrix
[[0.61 0.39]
 [0.18 0.82]]
```





In [37]:
```
print(classification_report(y_test.reshape(y_test.shape[0]  ,1),y_hat_3.round
```

```
              precision    recall  f1-score   support

           0       0.77      0.61      0.68       623
           1       0.68      0.82      0.74       624

    accuracy                           0.71      1247
   macro avg       0.72      0.71      0.71      1247
weighted avg       0.72      0.71      0.71      1247
```

In [39]:
```
from joblib import dump, load
dump(clf, 'Prediction/Ridge-FirePredModel.joblib')
```

Out[39]:
```
['Prediction/Ridge-FirePredModel.joblib']
```

NN

In [37]:
```
seed = 7
np.random.seed(seed)
# create model
def create_model():
    model = Sequential()
    model.add(Dense(500, input_dim=191, kernel_initializer='normal', activati
    model.add(Dropout(0.2))
    model.add(Dense(250, kernel_initializer='normal', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc
    return model
model = KerasClassifier(build_fn=create_model, verbose=1)
```

In [38]:
```
batch_size = [5]
epochs = [100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1,cv=5)
```

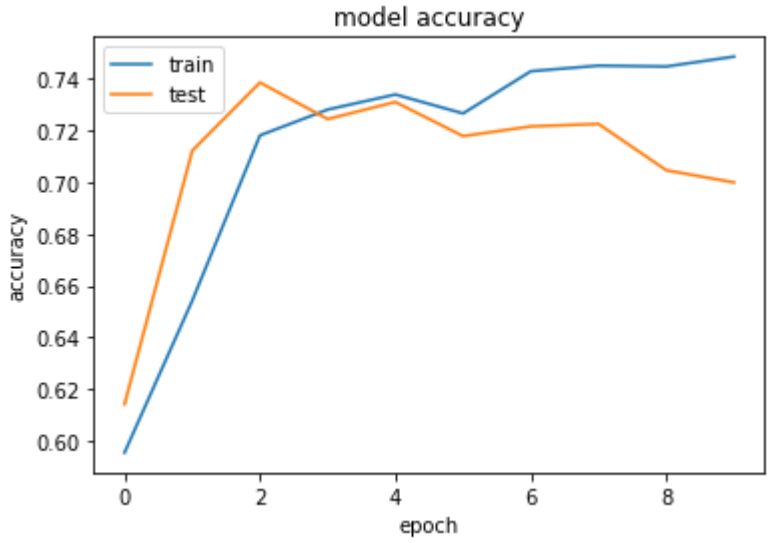
In [39]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,stratify=y
```

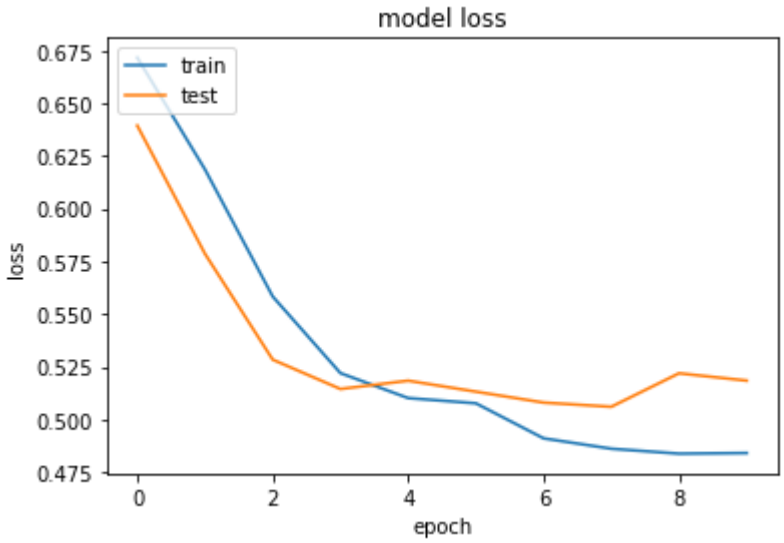
In [40]:
```
model = KerasClassifier(build_fn=create_model, verbose=1)
history = model.fit(X_train, y_train,epochs =10,batch_size=500 ,verbose=1,val
```

```
Train on 6001 samples, validate on 1060 samples
Epoch 1/10
6001/6001 [==============================] - 1s 182us/step - loss: 0.6716 -
accuracy: 0.5954 - val_loss: 0.6394 - val_accuracy: 0.6142
Epoch 2/10
6001/6001 [==============================] - 0s 74us/step - loss: 0.6186 - a
ccuracy: 0.6542 - val_loss: 0.5786 - val_accuracy: 0.7123
Epoch 3/10
6001/6001 [==============================] - 0s 76us/step - loss: 0.5583 - a
ccuracy: 0.7182 - val_loss: 0.5284 - val_accuracy: 0.7387
Epoch 4/10
6001/6001 [==============================] - 0s 76us/step - loss: 0.5220 - a
ccuracy: 0.7282 - val_loss: 0.5145 - val_accuracy: 0.7245
Epoch 5/10
6001/6001 [==============================] - 0s 78us/step - loss: 0.5102 - a
ccuracy: 0.7340 - val_loss: 0.5184 - val_accuracy: 0.7311
Epoch 6/10
6001/6001 [==============================] - 0s 79us/step - loss: 0.5077 - a
ccuracy: 0.7267 - val_loss: 0.5132 - val_accuracy: 0.7179
Epoch 7/10
6001/6001 [==============================] - 0s 81us/step - loss: 0.4911 - a
ccuracy: 0.7430 - val_loss: 0.5080 - val_accuracy: 0.7217
Epoch 8/10
6001/6001 [==============================] - 0s 79us/step - loss: 0.4861 - a
ccuracy: 0.7452 - val_loss: 0.5060 - val_accuracy: 0.7226
Epoch 9/10
6001/6001 [==============================] - 0s 72us/step - loss: 0.4838 - a
ccuracy: 0.7449 - val_loss: 0.5219 - val_accuracy: 0.7047
Epoch 10/10
6001/6001 [==============================] - 0s 74us/step - loss: 0.4841 - a
ccuracy: 0.7487 - val_loss: 0.5184 - val_accuracy: 0.7000
```

In [41]:
```
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

print('Max Epochs: %s \nMinimum Loss: %s' % (str(np.argmin(history.history['v
MaxEpochs = np.argmin(history.history['val_loss'])
MaxAcc = np.argmax(history.history['val_accuracy'])
print('Max Epochs: %s \nMax Accuracy: %s' % (str(np.argmax(history.history['v
```

```
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```





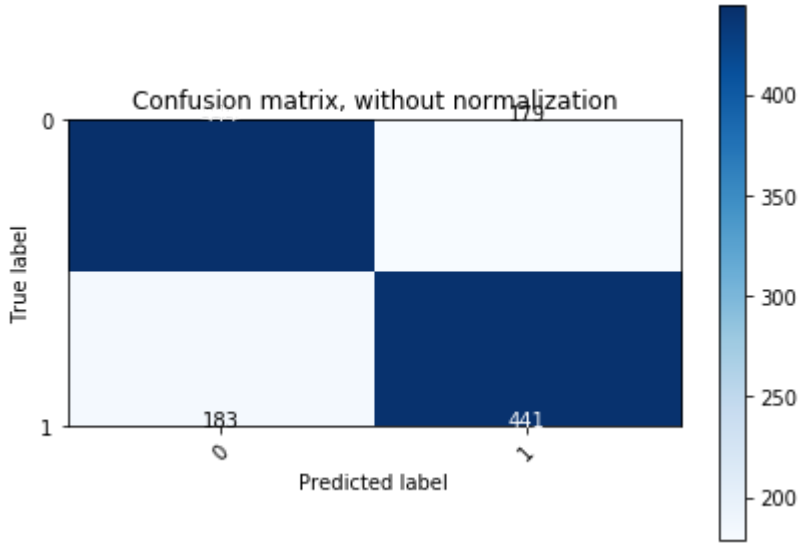

```
Max Epochs: 7
Minimum Loss: 0.5060371981476838
Max Epochs: 2
Max Accuracy: 0.7386792302131653
```

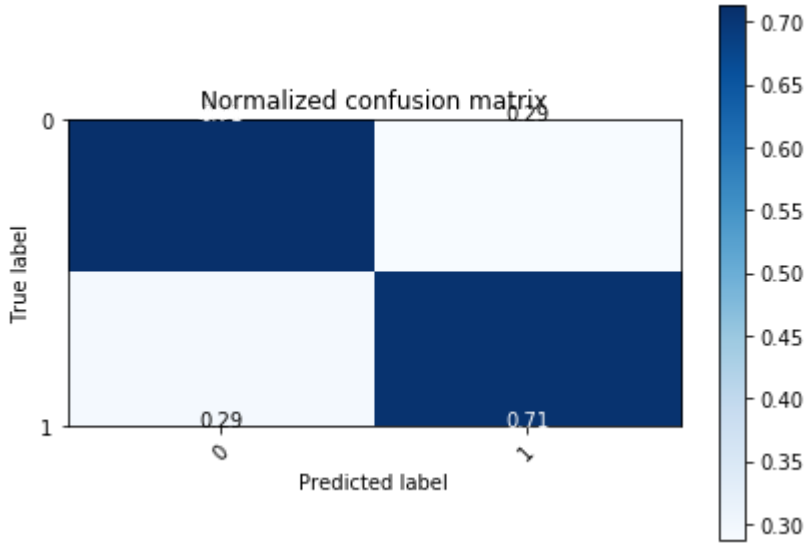
In [42]:
```
y_hat_2 = model.predict(X_test)
```

```
1247/1247 [==============================] - 0s 79us/step
```

In [43]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
corr = confusion_matrix(y_test.reshape(y_test.shape[0]  ,1),y_hat_2.round())
np.set_printoptions(precision=2)
class_names = [0,1]
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(corr, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(corr, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[444 179]
 [183 441]]
Normalized confusion matrix
[[0.71 0.29]
 [0.29 0.71]]
```





In [44]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test.reshape(y_test.shape[0]  ,1),y_hat_2.round
```

```
              precision    recall  f1-score   support

           0       0.71      0.71      0.71       623
           1       0.71      0.71      0.71       624

    accuracy                           0.71      1247
   macro avg       0.71      0.71      0.71      1247
weighted avg       0.71      0.71      0.71      1247
```

LSTM Time Series

In [ ]:
```

```