In [103]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve,roc_auc_score,auc
from tqdm import tqdm_notebook as tqdm
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from scipy.sparse import hstack,csr_matrix
import warnings
warnings.filterwarnings('ignore')
```

1. Data Loading & Preprocessing

In [104]:
```python
data=pd.read_csv('/home/sathish/Desktop/appliedai/Donors_choose_preprocessed_data.csv',na_filter=False)
data=data.sample(50000,random_state=42)

y=data['project_is_approved'].values
x=data.drop(['project_is_approved'],axis=1)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,stratify=y)
```

In [105]:
```python
essays=x_train['essay']
all_essay_text=' '.join(essays)
words=all_essay_text.split(" ")
import pickle
with open('PCA/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

from collections import Counter
counts=Counter(words)
vocab=sorted(counts,key=counts.get,reverse=True)
vocab=glove_words.intersection(vocab)
int_to_vocab = {ii:word  for ii, word in enumerate(vocab,1)}
vocab_to_int={word:ii  for ii, word in enumerate(vocab,1)}
train_essay_ints = []
for essay in essays:
    train_essay_ints.append([vocab_to_int[word] for word in essay.split() if word in vocab])

essays=x_test['essay']
test_essay_ints = []
for essay in essays:
    test_essay_ints.append([vocab_to_int[word] for word in essay.split() if word in vocab])
```

In [106]:
```python
#for fixed length input
def pad_features(essay_ints, seq_length):

    features = np.zeros((len(essay_ints), seq_length), dtype=int)


    for i, row in enumerate(essay_ints):
        if len(row)==0:
            continue
        features[i, -len(row):] = np.array(row)[:seq_length]

    return features
```

In [107]:
```python
seq_length = 50

train_features = pad_features(train_essay_ints, seq_length=seq_length)
test_features=pad_features(test_essay_ints, seq_length=seq_length)


assert len(train_features)==len(train_essay_ints)
assert len(train_features[0])==seq_length

print(train_features[:5,:10])
```

```
[[  285 24812 32699 17056 14253  7456   998  9587 18102 11903]
 [17334 24812 26396 22654 12924 26306 17251 31885 31485 12924]
 [23720  1193 25451 26396  1382  5876 16845 24812 21758 25336]
 [ 2251 22654 25336 19575 14225 26396  3942 26967 18312  5920]
 [16688 31813 32828  2276 18466 16904  5615 24812 16698  4066]]
```

In [108]:
```python
weight_matrix=np.random.normal(size=(len(vocab)+1,300))
for i in range(1,len(vocab)+1):
    weight_matrix[i]=model[int_to_vocab[i]]
```

In [109]:
```python
values=x_train['school_state']
counts=Counter(values)
vocab=sorted(counts,key=counts.get,reverse=True)
vocab_to_int = {word: ii for ii, word in enumerate(vocab,1)}
train_school_state_ints=[vocab_to_int[word] for word in values]
values=x_test['school_state']
test_school_state_ints=[vocab_to_int[word] if word in vocab else 0 for word in values]
len_ss_vocab=len(vocab)+1
```

In [110]:
```python
values=x_train['project_grade_category']
counts=Counter(values)
vocab=sorted(counts,key=counts.get,reverse=True)
vocab_to_int = {word: ii for ii, word in enumerate(vocab,1)}
train_grades_ints=[vocab_to_int[word] for word in values]
values=x_test['project_grade_category']
test_grades_ints=[vocab_to_int[word] if word in vocab else 0 for word in values]
len_pg_vocab=len(vocab)+1
```

In [111]:
```python
values=x_train['project_subject_categories']
counts=Counter(values)
vocab=sorted(counts,key=counts.get,reverse=True)
vocab_to_int = {word: ii for ii, word in enumerate(vocab,1)}
train_categories_ints=[vocab_to_int[word] for word in values]
values=x_test['project_subject_categories']
test_categories_ints=[vocab_to_int[word] if word in vocab else 0 for word in values]
len_sc_vocab=len(vocab)+1
```

In [112]:
```python
values=x_train['project_subject_subcategories']
counts=Counter(values)
vocab=sorted(counts,key=counts.get,reverse=True)
vocab_to_int = {word: ii for ii, word in enumerate(vocab,1)}
train_sub_categories_ints=[vocab_to_int[word] for word in values]
values=x_test['project_subject_subcategories']
test_sub_categories_ints=[vocab_to_int[word] if word in vocab else 0 for word in values]
len_ssc_vocab=len(vocab)+1
```

In [113]:
```python
values=x_train['teacher_prefix']
counts=Counter(values)
vocab=sorted(counts,key=counts.get,reverse=True)
vocab_to_int = {word: ii for ii, word in enumerate(vocab,1)}
train_teacher_ints=[vocab_to_int[word] for word in values]
values=x_test['teacher_prefix']
test_teacher_ints=[vocab_to_int[word] if word in vocab else 0 for word in values]
len_tp_vocab=len(vocab)+1
```

In [114]:
```python
def check_digits(value):
    for word in value.split():
        if word.isdigit():
            return 1
    return 0

standardscalar=StandardScaler()#standardizing data
train_prev_projects=np.array(x_train['teacher_number_of_previously_posted_projects']).reshape(-1,1)
train_prev_projects=standardscalar.fit_transform(train_prev_projects)
test_prev_projects=np.array(x_test['teacher_number_of_previously_posted_projects']).reshape(-1,1)
test_prev_projects=standardscalar.transform(test_prev_projects)


train_price=np.array(x_train['price']).reshape(-1,1)
train_price=standardscalar.fit_transform(train_price)
test_price=np.array(x_test['price']).reshape(-1,1)
test_price=standardscalar.transform(test_price)


train_quantity=np.array(x_train['quantity']).reshape(-1,1)
train_quantity=standardscalar.fit_transform(train_quantity)
test_quantity=np.array(x_test['quantity']).reshape(-1,1)
test_quantity=standardscalar.transform(test_quantity
                                       )
train_summary_contains_digits=np.array(x_train['project_resource_summary'].map(check_digits)).reshape(-1,1)
test_summary_contains_digits=np.array(x_test['project_resource_summary'].map(check_digits)).reshape(-1,1)

train_remain_features=np.concatenate((train_prev_projects,train_price,train_quantity,train_summary_contains_digits),axis=1)
test_remain_features=np.concatenate((test_prev_projects,test_price,test_quantity,test_summary_contains_digits),axis=1)
```

2.LSTM Model Building (model-1)

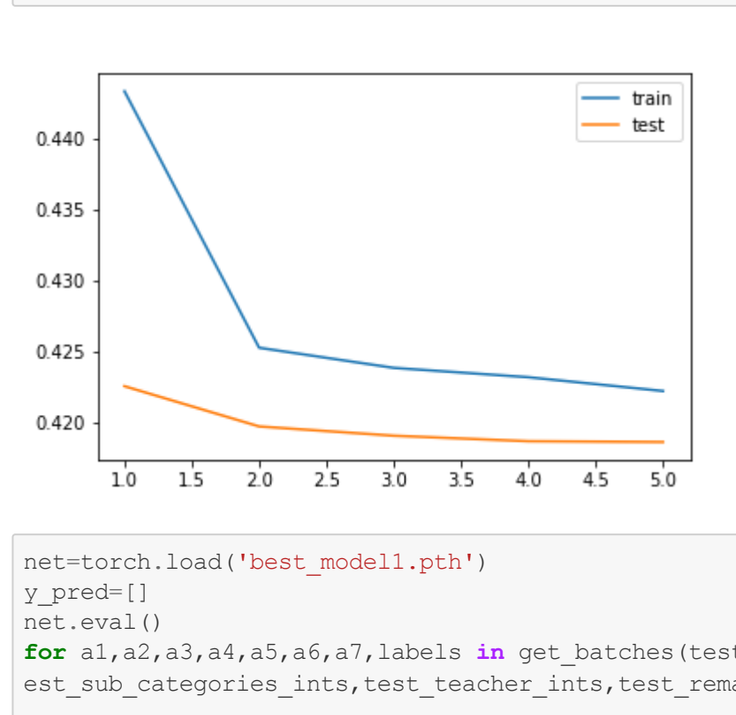
In [115]:
```python
def create_emb_layer(weights_matrix, non_trainable=False):
    #to create embedding layer with our pre trained weights(in this case word vectors)
    num_embeddings, embedding_dim = weights_matrix.size()
    emb_layer = nn.Embedding(num_embeddings, embedding_dim)
    emb_layer.load_state_dict({'weight': weights_matrix})
    if non_trainable:
        emb_layer.weight.requires_grad = False

    return emb_layer, num_embeddings, embedding_dim

class Net(nn.Module):
    def __init__(self, weights_matrix,l1,l2,l3,l4,l5,hidden_size,num_layers):
        super().__init__()
        self.embedding, num_embeddings, embedding_dim = create_emb_layer(weights_matrix,True)
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.emb1=nn.Embedding(l1,30)
        self.emb2=nn.Embedding(l2,10)
        self.emb3=nn.Embedding(l3,30)
        self.emb4=nn.Embedding(l4,30)
        self.emb5=nn.Embedding(l5,10)
        self.dense=nn.Linear(4,20)
        self.dense2=nn.Linear(280,50)
        self.dense3=nn.Linear(50,30)
        self.dense4=nn.Linear(30,1)
        self.dropout=nn.Dropout(0.2)
        self.sigmoid=nn.Sigmoid()
        self.lstm = nn.LSTM(embedding_dim, hidden_size, num_layers, batch_first=True)

    def forward(self,inp,x1,x2,x3,x4,x5,rem,hidden):
        inp,hidden=self.lstm(self.embedding(inp),hidden)
        inp=inp[:,-1,:]
        x1=self.emb1(x1)
        x2=self.emb2(x2)
        x3=self.emb3(x3)
        x4=self.emb4(x4)
        x5=self.emb5(x5)
        rem=self.dense(rem)
        lstm_out = inp.contiguous().view(-1, self.hidden_size) # stack up lstm outputs
        features=torch.cat((inp,x1,x2,x3,x4,x5,rem),1)
        features=self.dropout(features)
        features=self.dense2(features)
        features=self.dropout(features)
        features=self.dense3(features)
        features=self.dense4(features)
        features=self.sigmoid(features)
        return features


    def init_hidden(self, batch_size):
        hidden=torch.zeros(self.num_layers,batch_size,self.hidden_size),torch.zeros(self.num_layers,batch_size,self.hidden_si
ze)
        return hidden
```

In [116]:
```python
batch_size=32
def get_batches(inp,x1,x2,x3,x4,x5,rem,y):
    total_length=len(inp)
    num_batch=total_length//batch_size
    total_length=num_batch*batch_size
    inp=inp[:total_length]
    x1=x1[:total_length]
    x2=x2[:total_length]
    x3=x3[:total_length]
    x4=x4[:total_length]
    x5=x5[:total_length]
    rem=rem[:total_length]
    y=y[:total_length]

    for i in range(num_batch):
        start=i*batch_size
        end=start+batch_size
        yield inp[start:end],x1[start:end],x2[start:end],x3[start:end],x4[start:end],x5[start:end],rem[start:end],y[start:end
]
```

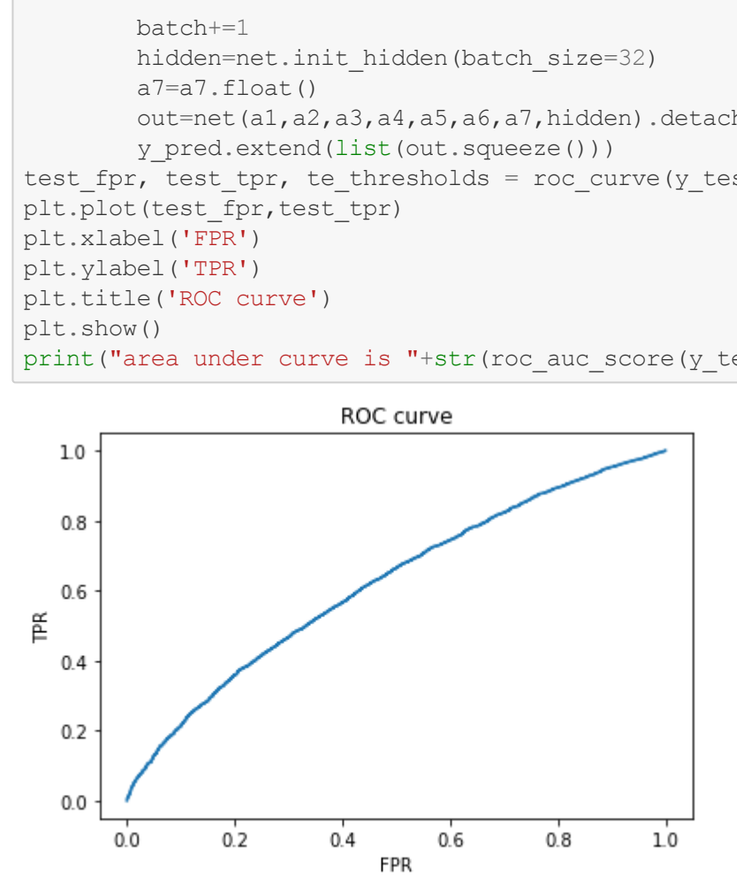
In [117]:
```python
train_essay_ints=torch.from_numpy(train_features)
train_school_state_ints=torch.from_numpy(np.array(train_school_state_ints))
train_grades_ints=torch.from_numpy(np.array(train_grades_ints))
train_categories_ints=torch.from_numpy(np.array(train_categories_ints))
train_sub_categories_ints=torch.from_numpy(np.array(train_sub_categories_ints))
train_teacher_ints=torch.from_numpy(np.array(train_teacher_ints))
train_remain_features=torch.from_numpy(train_remain_features)
train_expected=torch.from_numpy(y_train)


test_essay_ints=torch.from_numpy(test_features)
test_school_state_ints=torch.from_numpy(np.array(test_school_state_ints))
test_grades_ints=torch.from_numpy(np.array(test_grades_ints))
test_categories_ints=torch.from_numpy(np.array(test_categories_ints))
test_sub_categories_ints=torch.from_numpy(np.array(test_sub_categories_ints))
test_teacher_ints=torch.from_numpy(np.array(test_teacher_ints))
test_remain_features=torch.from_numpy(test_remain_features)
test_expected=torch.from_numpy(y_test)
```

In [118]:
```python
net=Net(torch.from_numpy(weight_matrix),len_ss_vocab,len_pg_vocab,len_sc_vocab,len_ssc_vocab,len_tp_vocab,150,2)
lr=0.001
criterion = nn.BCELoss()
optimizer = torch.optim.SGD(net.parameters(), lr=lr,momentum=0.9)
```

In [119]:
```python
epochs=5
epoch_train_loss=[]
epoch_test_loss=[]
best_loss=np.inf
for epoch in tqdm(range(epochs)):
    train_loss=0
    test_loss=0
    batch=0
    net.train()
    for a1,a2,a3,a4,a5,a6,a7,labels in get_batches(train_essay_ints,train_school_state_ints,train_grades_ints,train_categorie
s_ints,train_sub_categories_ints,train_teacher_ints,train_remain_features,train_expected):
        batch+=1
        hidden=net.init_hidden(batch_size=32)
        net.zero_grad()
        a7=a7.float()
        out=net(a1,a2,a3,a4,a5,a6,a7,hidden)
        loss = criterion(out.squeeze(), labels.float())
        loss.backward()
        optimizer.step()
        train_loss+=loss.item()
    epoch_train_loss.append(train_loss/batch)
    net.eval()
    batch=0
    for a1,a2,a3,a4,a5,a6,a7,labels in get_batches(test_essay_ints,test_school_state_ints,test_grades_ints,test_categories_in
ts,test_sub_categories_ints,test_teacher_ints,test_remain_features,test_expected):
        batch+=1
        hidden=net.init_hidden(batch_size=32)
        a7=a7.float()
        out=net(a1,a2,a3,a4,a5,a6,a7,hidden)
        loss = criterion(out.squeeze(), labels.float())
        test_loss+=loss.item()
    if best_loss>test_loss:
            torch.save(net,'best_model1.pth')
            best_loss=test_loss
    epoch_test_loss.append(test_loss/batch)


plt.plot(range(1,epochs+1),epoch_train_loss,label='train')
plt.plot(range(1,epochs+1),epoch_test_loss,label='test')
plt.legend()
plt.show()
```



In [120]:
```python
net=torch.load('best_model1.pth')
y_pred=[]
net.eval()
for a1,a2,a3,a4,a5,a6,a7,labels in get_batches(test_essay_ints,test_school_state_ints,test_grades_ints,test_categories_ints,t
est_sub_categories_ints,test_teacher_ints,test_remain_features,test_expected):

        batch+=1
        hidden=net.init_hidden(batch_size=32)
        a7=a7.float()
        out=net(a1,a2,a3,a4,a5,a6,a7,hidden).detach().numpy()
        y_pred.extend(list(out.squeeze()))
test_fpr, test_tpr, te_thresholds = roc_curve(y_test[:len(y_pred)], y_pred)
plt.plot(test_fpr,test_tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.show()
print("area under curve is "+str(roc_auc_score(y_test[:len(y_pred)],y_pred)))
```

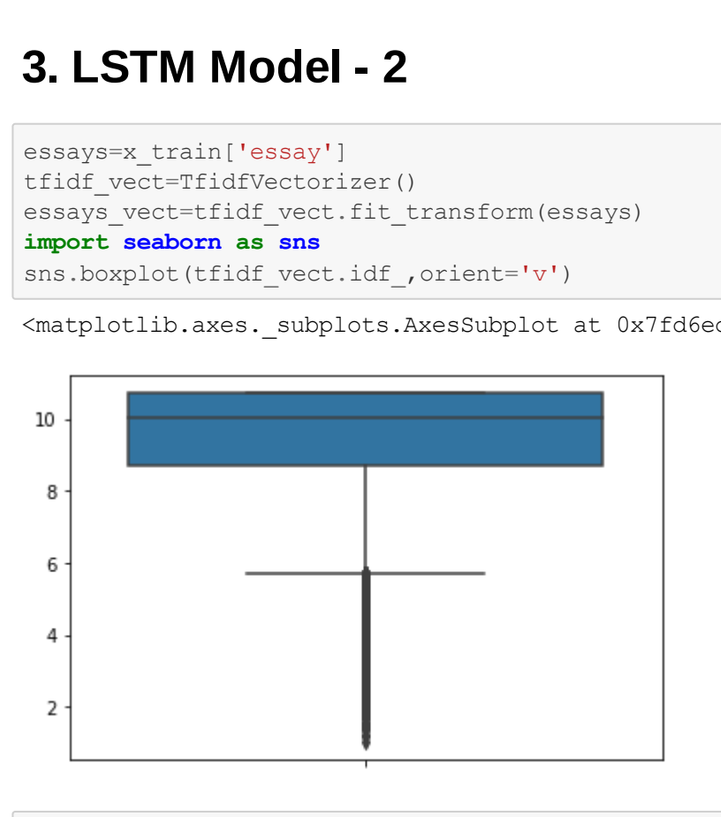



```
area under curve is 0.6215763115341881
```

3. LSTM Model - 2

In [121]:
```python
essays=x_train['essay']
tfidf_vect=TfidfVectorizer()
essays_vect=tfidf_vect.fit_transform(essays)
import seaborn as sns
sns.boxplot(tfidf_vect.idf_,orient='v')
```

Out[121]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd6edbe5278>
```



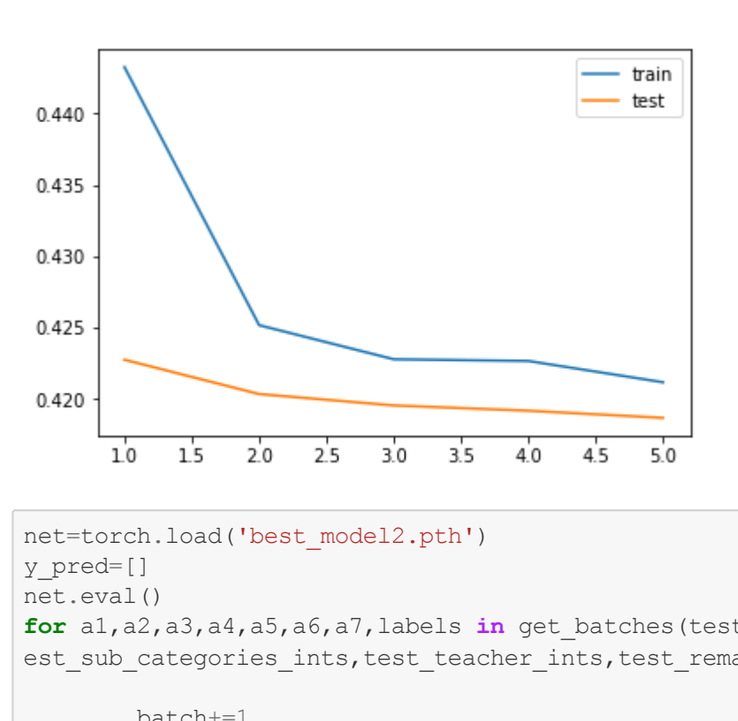
In [122]:
```python
first_quartile=np.percentile(tfidf_vect.idf_,25);third_quartile=np.percentile(tfidf_vect.idf_,75)
vocab_dict=dict(zip(tfidf_vect.get_feature_names(),tfidf_vect.idf_))
vocab=[word for word,idf in vocab_dict.items() if idf>first_quartile and idf<third_quartile ]
```

In [123]:
```python
vocab=glove_words.intersection(vocab)
int_to_vocab = {ii:word  for ii, word in enumerate(vocab,1)}
vocab_to_int={word:ii  for ii, word in enumerate(vocab,1)}
essays=x_train['essay']


train_essay_ints=[]
for essay in essays:
    train_essay_ints.append([vocab_to_int[word] for word in essay.split() if word in vocab])

essays=x_test['essay']
test_essay_ints=[]
for essay in essays:
    test_essay_ints.append([vocab_to_int[word] for word in essay.split() if word in vocab])
```

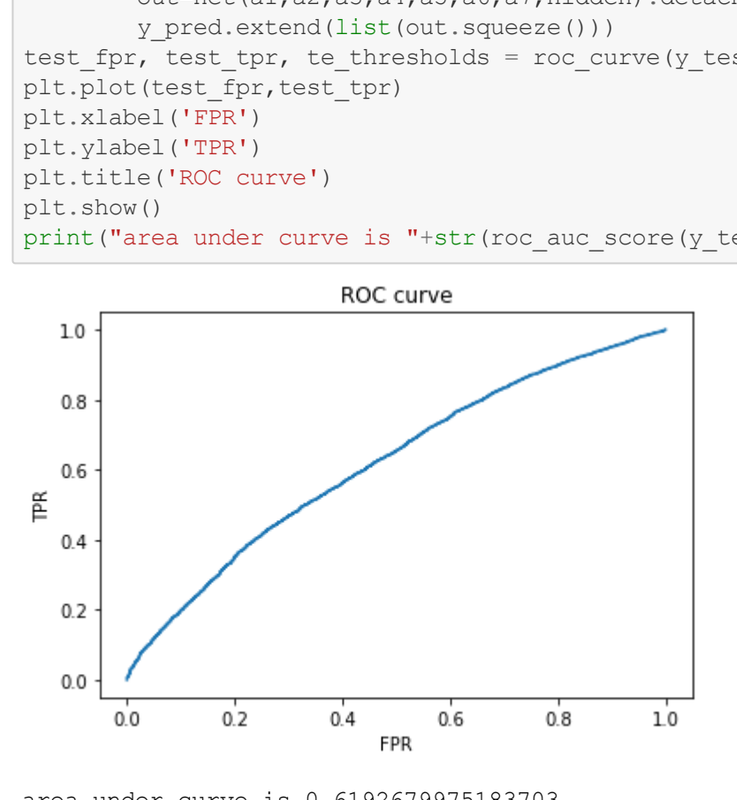
In [124]:
```python
seq_length = 50

train_features = pad_features(train_essay_ints, seq_length=seq_length)
test_features=pad_features(test_essay_ints, seq_length=seq_length)
weight_matrix=np.random.normal(size=(len(vocab)+1,300))
for i in range(1,len(vocab)+1):
    weight_matrix[i]=model[int_to_vocab[i]]

test_essay_ints=torch.from_numpy(test_features)
train_essay_ints=torch.from_numpy(train_features)
```

In [125]:
```python
net=Net(torch.from_numpy(weight_matrix),len_ss_vocab,len_pg_vocab,len_sc_vocab,len_ssc_vocab,len_tp_vocab,150,2)
lr=0.001
criterion = nn.BCELoss()
optimizer = torch.optim.SGD(net.parameters(), lr=lr,momentum=0.9)
```

In [126]:
```python
epoch_train_loss=[]
epoch_test_loss=[]
best_loss=np.inf
for epooch in tqdm(range(epochs)):
    train_loss=0
    test_loss=0
    batch=0
    net.train()
    for a1,a2,a3,a4,a5,a6,a7,labels in get_batches(train_essay_ints,train_school_state_ints,train_grades_ints,train_categorie
s_ints,train_sub_categories_ints,train_teacher_ints,train_remain_features,train_expected):
        batch+=1
        hidden=net.init_hidden(batch_size=32)
        net.zero_grad()
        a7=a7.float()
        out=net(a1,a2,a3,a4,a5,a6,a7,hidden)
        loss = criterion(out.squeeze(), labels.float())
        loss.backward()
        optimizer.step()
        train_loss+=loss.item()
    epoch_train_loss.append(train_loss/batch)
    net.eval()
    batch=0
    for a1,a2,a3,a4,a5,a6,a7,labels in get_batches(test_essay_ints,test_school_state_ints,test_grades_ints,test_categories_in
ts,test_sub_categories_ints,test_teacher_ints,test_remain_features,test_expected):
        batch+=1
        hidden=net.init_hidden(batch_size=32)
        a7=a7.float()
        out=net(a1,a2,a3,a4,a5,a6,a7,hidden)
        loss = criterion(out.squeeze(), labels.float())
        test_loss+=loss.item()
    if best_loss>test_loss:
            torch.save(net,'best_model2.pth')
            best_loss=test_loss
    epoch_test_loss.append(test_loss/batch)


plt.plot(range(1,epochs+1),epoch_train_loss,label='train')
plt.plot(range(1,epochs+1),epoch_test_loss,label='test')
plt.legend()
plt.show()
```



In [127]:
```python
net=torch.load('best_model2.pth')
y_pred=[]
net.eval()
for a1,a2,a3,a4,a5,a6,a7,labels in get_batches(test_essay_ints,test_school_state_ints,test_grades_ints,test_categories_ints,t
est_sub_categories_ints,test_teacher_ints,test_remain_features,test_expected):

        batch+=1
        hidden=net.init_hidden(batch_size=32)
        a7=a7.float()
        out=net(a1,a2,a3,a4,a5,a6,a7,hidden).detach().numpy()
        y_pred.extend(list(out.squeeze()))
test_fpr, test_tpr, te_thresholds = roc_curve(y_test[:len(y_pred)], y_pred)
plt.plot(test_fpr,test_tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.show()
print("area under curve is "+str(roc_auc_score(y_test[:len(y_pred)],y_pred)))
```




```
area under curve is 0.6192679975183703
```

4. LSTM Model- 3

In [128]:
```python
essays=x_train['essay']
all_essay_text=' '.join(essays)
words=all_essay_text.split(" ")
import pickle
with open('PCA/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

from collections import Counter
counts=Counter(words)
vocab=sorted(counts,key=counts.get,reverse=True)
vocab=glove_words.intersection(vocab)
int_to_vocab = {ii:word  for ii, word in enumerate(vocab,1)}
vocab_to_int={word:ii  for ii, word in enumerate(vocab,1)}
train_essay_ints = []
for essay in essays:
    train_essay_ints.append([vocab_to_int[word] for word in essay.split() if word in vocab])

essays=x_test['essay']
test_essay_ints = []
for essay in essays:
    test_essay_ints.append([vocab_to_int[word] for word in essay.split() if word in vocab])
```

In [129]:
```python
def pad_features(essay_ints, seq_length):

    features = np.zeros((len(essay_ints), seq_length), dtype=int)


    for i, row in enumerate(essay_ints):
        if len(row)==0:
            continue
        features[i, -len(row):] = np.array(row)[:seq_length]

    return features
```

In [130]:
```python
seq_length = 50

train_features = pad_features(train_essay_ints, seq_length=seq_length)
test_features=pad_features(test_essay_ints, seq_length=seq_length)


assert len(train_features)==len(train_essay_ints)
assert len(train_features[0])==seq_length

print(train_features[:5,:10])
```

```
[[  285 24812 32699 17056 14253  7456   998  9587 18102 11903]
 [17334 24812 26396 22654 12924 26306 17251 31885 31485 12924]
 [23720  1193 25451 26396  1382  5876 16845 24812 21758 25336]
 [ 2251 22654 25336 19575 14225 26396  3942 26967 18312  5920]
 [16688 31813 32828  2276 18466 16904  5615 24812 16698  4066]]
```

In [131]:
```python
weight_matrix=np.random.normal(size=(len(vocab)+1,300))
for i in range(1,len(vocab)+1):
    weight_matrix[i]=model[int_to_vocab[i]]
```

In [132]:
```python
vectorizer=CountVectorizer()
x_train_state_onehot=vectorizer.fit_transform(x_train['school_state'].values)
x_test_state_onehot=vectorizer.transform(x_test['school_state'].values)


x_train_tprefix_onehot=vectorizer.fit_transform(x_train['teacher_prefix'].values)
x_test_tprefix_onehot=vectorizer.transform(x_test['teacher_prefix'].values)


x_train_grade_onehot=vectorizer.fit_transform(x_train['project_grade_category'].values)
x_test_grade_onehot=vectorizer.transform(x_test['project_grade_category'].values)


x_train_cat_onehot=vectorizer.fit_transform(x_train['project_subject_categories'].values)
x_test_cat_onehot=vectorizer.transform(x_test['project_subject_categories'].values)


x_train_subcat_onehot=vectorizer.fit_transform(x_train['project_subject_subcategories'].values)
x_test_subcat_onehot=vectorizer.transform(x_test['project_subject_subcategories'].values)

#all categrocial features
x_train_cat=hstack((x_train_state_onehot,x_train_tprefix_onehot,x_train_grade_onehot,x_train_cat_onehot,x_train_subcat_onehot
))
x_test_cat=hstack((x_test_state_onehot,x_test_tprefix_onehot,x_test_grade_onehot,x_test_cat_onehot,x_test_subcat_onehot))
```

In [133]:
```python
standardscaler=StandardScaler()
x_train_price_norm=standardscaler.fit_transform(x_train['price'].values.reshape(-1,1))
x_test_price_norm=standardscaler.transform(x_test['price'].values.reshape(-1,1))

x_train_quantity_norm=standardscaler.fit_transform(x_train['quantity'].values.reshape(-1,1))
x_test_quantity_norm=standardscaler.transform(x_test['quantity'].values.reshape(-1,1))

x_train_prev_projects_norm=standardscaler.fit_transform(x_train['teacher_number_of_previously_posted_projects'].values.reshap
e(-1,1))
x_test_prev_projects_norm=standardscaler.transform(x_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1
))

#all numerical features
x_train_num=hstack((csr_matrix(x_train_price_norm),x_train_quantity_norm,x_train_prev_projects_norm))
x_test_num=hstack((csr_matrix(x_test_price_norm),x_test_quantity_norm,x_test_prev_projects_norm))
```

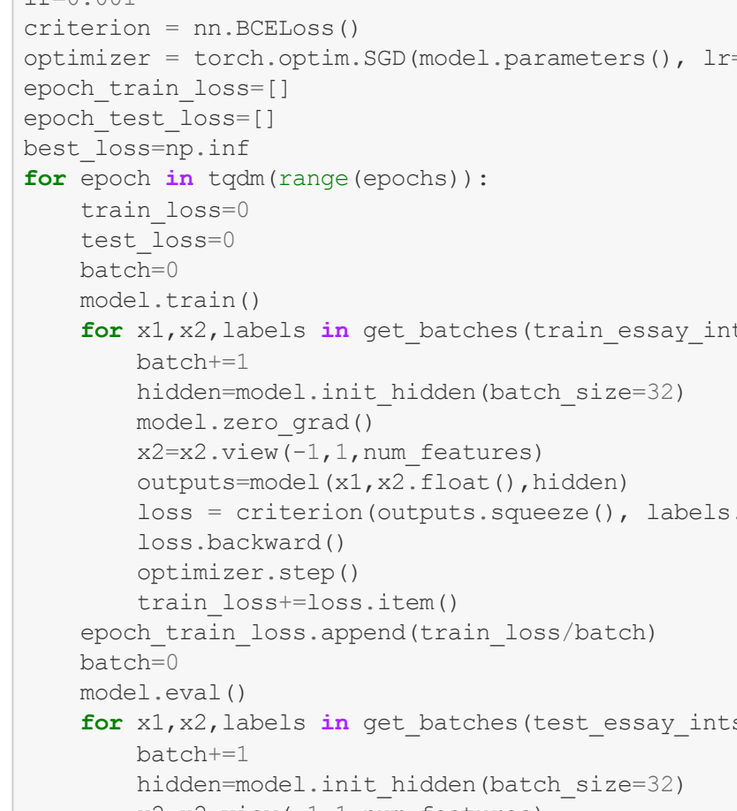
In [134]:
```python
final_train=hstack((x_train_cat,x_train_num)).tocsr()
final_test=hstack((x_test_cat,x_test_num)).tocsr()
num_features=final_train.shape[1]
```

In [135]:
```python
weight_matrix=torch.from_numpy(weight_matrix)
train_essay_ints=torch.from_numpy(train_features)
test_essay_ints=torch.from_numpy(test_features)
final_train=torch.from_numpy(final_train.toarray())
final_test=torch.from_numpy(final_test.toarray())
```

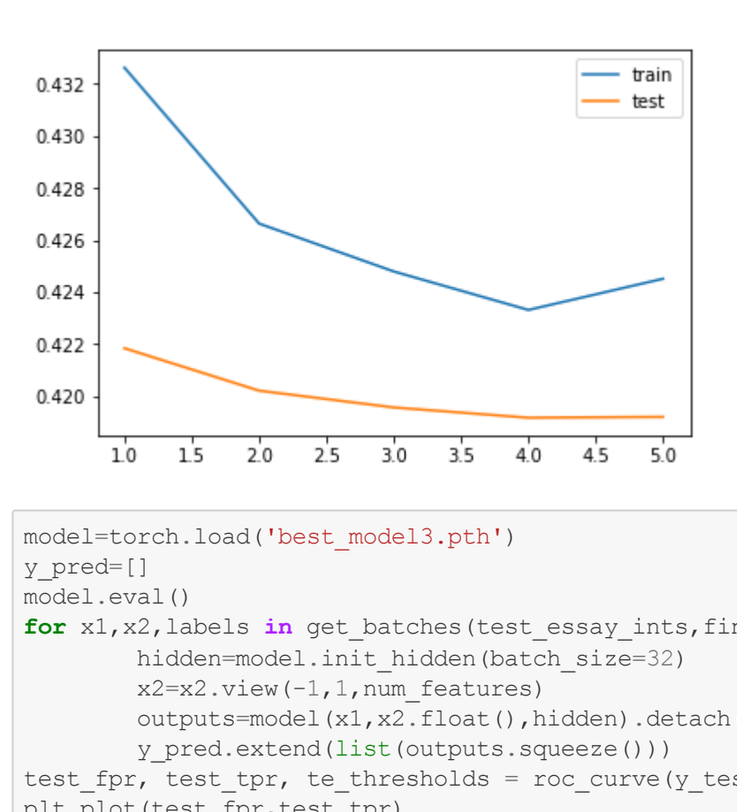
In [136]:
```python
def create_emb_layer(weights_matrix, non_trainable=False):

    num_embeddings, embedding_dim = weights_matrix.size()
    emb_layer = nn.Embedding(num_embeddings, embedding_dim)
    emb_layer.load_state_dict({'weight': weights_matrix})
    if non_trainable:
        emb_layer.weight.requires_grad = False

    return emb_layer, num_embeddings, embedding_dim

class Model(nn.Module):
    def __init__(self, weights_matrix,hidden_size,num_layers):
        super().__init__()
        self.embedding, num_embeddings, embedding_dim = create_emb_layer(weights_matrix, True)
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.conv1=nn.Conv1d(1,200,3,padding=1)
        self.dense2=nn.Linear(200*num_features+hidden_size,400)
        self.dense3=nn.Linear(400,100)
        self.dense4=nn.Linear(100,30)
        self.dense=nn.Linear(30,1)
        self.dropout=nn.Dropout(0.5)
        self.sigmoid=nn.Sigmoid()
        self.lstm = nn.LSTM(embedding_dim, hidden_size, num_layers, batch_first=True)

    def forward(self,x1,x2,hidden):
        lstm_out,hidden=self.lstm(self.embedding(x1),hidden)
        lstm_out=lstm_out[:,-1,:] #for last time step output only
        lstm_out = lstm_out.contiguous().view(-1, self.hidden_size) # stack up lstm outputs
        x2=self.conv1(x2)
        x2=x2.view(batch_size,-1)

        features=torch.cat((lstm_out,x2),1)
        features=self.dense2(features)
        features=self.dropout(features)
        features=self.dense3(features)
        features=self.dropout(features)
        features=self.dense4(features)
        features=self.dense(features)

        features=self.sigmoid(features)
        return features


    def init_hidden(self, batch_size):
        hidden=torch.zeros(self.num_layers,batch_size,self.hidden_size),torch.zeros(self.num_layers,batch_size,self.hidden_si
ze)
        return hidden
```

In [137]:
```python
batch_size=32
def get_batches(inp1,inp2,y):
    total_length=len(inp1)
    num_batch=total_length//batch_size
    total_length=num_batch*batch_size
    inp1=inp1[:total_length]
    inp2=inp2[:total_length]
    y=y[:total_length]
    for i in range(num_batch):
        start=i*batch_size
        end=start+batch_size
        yield inp1[start:end],inp2[start:end],y[start:end]
```

In [138]:
```python
model=Model(weight_matrix,150,2)
lr=0.001
criterion = nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr,momentum=0.9)
epoch_train_loss=[]
epoch_test_loss=[]
best_loss=np.inf
for epoch in tqdm(range(epochs)):
    train_loss=0
    test_loss=0
    batch=0
    model.train()
    for x1,x2,labels in get_batches(train_essay_ints,final_train,train_expected):
        batch+=1
        hidden=model.init_hidden(batch_size=32)
        model.zero_grad()
        x2=x2.view(-1,1,num_features)
        outputs=model(x1,x2.float(),hidden)
        loss = criterion(outputs.squeeze(), labels.float())
        loss.backward()
        optimizer.step()
        train_loss+=loss.item()
    epoch_train_loss.append(train_loss/batch)
    batch=0
    model.eval()
    for x1,x2,labels in get_batches(test_essay_ints,final_test,test_expected):
        batch+=1
        hidden=model.init_hidden(batch_size=32)
        x2=x2.view(-1,1,num_features)
        outputs=model(x1,x2.float(),hidden)
        loss = criterion(outputs.squeeze(), labels.float())
        test_loss+=loss.item()
    if best_loss>test_loss:
            torch.save(model,'best_model3.pth')
            best_loss=test_loss
    epoch_test_loss.append(test_loss/batch)

plt.plot(range(1,epochs+1),epoch_train_loss,label='train')
plt.plot(range(1,epochs+1),epoch_test_loss,label='test')
plt.legend()
plt.show()
```



In [139]:
```python
model=torch.load('best_model3.pth')
y_pred=[]
model.eval()
for x1,x2,labels in get_batches(test_essay_ints,final_test,test_expected):
        hidden=model.init_hidden(batch_size=32)
        x2=x2.view(-1,1,num_features)
        outputs=model(x1,x2.float(),hidden).detach().numpy()
        y_pred.extend(list(outputs.squeeze()))
test_fpr, test_tpr, te_thresholds = roc_curve(y_test[:len(y_pred)], y_pred)
plt.plot(test_fpr,test_tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.show()
print("area under curve is "+str(roc_auc_score(y_test[:len(y_pred)],y_pred)))
```




```
area under curve is 0.6226780397439795
```