In [1]:
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
%matplotlib inline
```

In [2]:
```
ls dataa
```

```
 Volume in drive C is OS
 Volume Serial Number is 385E-088B

 Directory of C:\Users\Rudra\Untitled Folder 5\Freeai\Face_recognition\cLaSS\module-2\dataa

06/22/2020  02:19 PM    <DIR>          .
06/22/2020  02:19 PM    <DIR>          ..
06/20/2020  10:48 PM    <DIR>          crop
06/21/2020  08:41 PM       436,662,338 data_10000_norm.npz
06/22/2020  02:19 PM         2,205,788 Data_pca_mean_50.pickle.npz
06/21/2020  08:10 PM        54,766,670 dataframe_images_100_100.pickle
06/20/2020  12:47 PM    <DIR>          female
03/22/2020  10:12 AM           930,127 haarcascade_frontalface_default.xml
06/20/2020  12:43 PM    <DIR>          male
06/22/2020  01:44 PM         4,082,046 pca_50.pickle
               5 File(s)    498,646,969 bytes
               5 Dir(s)  596,238,811,136 bytes free
```

In [5]:
```
# load the data

data = np.load('./dataa/Data_pca_mean_50.pickle.npz')
data.files
```

Out[5]:
```
['arr_0', 'arr_1', 'arr_2']
```

In [6]:
```
X = data['arr_0']
y = data['arr_1']
mean = data['arr_2']
```

In [7]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.2,stratify=y)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[7]:
```
((4366, 50), (1092, 50), (4366,), (1092,))
```

In [8]:
```
# training the model
from sklearn.svm import SVC
```

In [13]:
```
model = SVC(C=1.0,kernel='rbf',gamma=0.01,probability=True)
```

In [14]:
```
model.fit(x_train,y_train)
print('trained successfully')
```

```
trained successfully
```

In [15]:
```
# score
model.score(x_train,y_train)
```

Out[15]:
```
0.8547869903802107
```

In [16]:
```
# score for test data
model.score(x_test,y_test)
```

Out[16]:
```
0.804945054945055
```

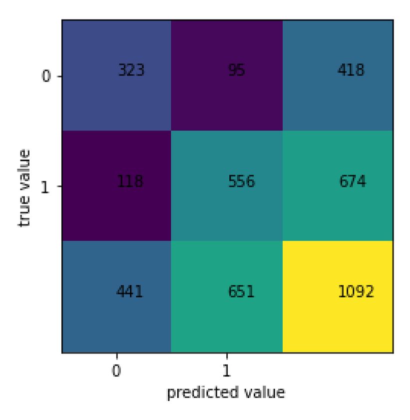
Evaluation

In [18]:
```
# confusion matrix
from sklearn import metrics
```

In [19]:
```
y_pred = model.predict(x_test)
y_prob = model.predict_proba(x_test)
```

In [39]:
```
cm = metrics.confusion_matrix(y_test,y_pred)

cm = np.concatenate((cm,cm.sum(axis=0).reshape(1,-1)),axis=0)
cm = np.concatenate((cm,cm.sum(axis=1).reshape(-1,1)),axis=1)
cm

plt.imshow(cm)
for i in range(3):
    for j in range(3):
        plt.text(i,j,'%d'%cm[i,j])

plt.xticks([0,1])
plt.yticks([0,1])
plt.xlabel('predicted value')
plt.ylabel('true value')
```

Out[39]:
```
Text(0, 0.5, 'true value')
```



In [42]:
```
cr = metrics.classification_report(y_test,y_pred,target_names=['male','female'],output_dict=
True)
pd.DataFrame(cr).T
```

Out[42]:

	precision	recall	f1-score	support
male	0.772727	0.732426	0.752037	441.000000
female	0.824926	0.854071	0.839245	651.000000
accuracy	0.804945	0.804945	0.804945	0.804945
macro avg	0.798827	0.793248	0.795641	1092.000000
weighted avg	0.803846	0.804945	0.804027	1092.000000

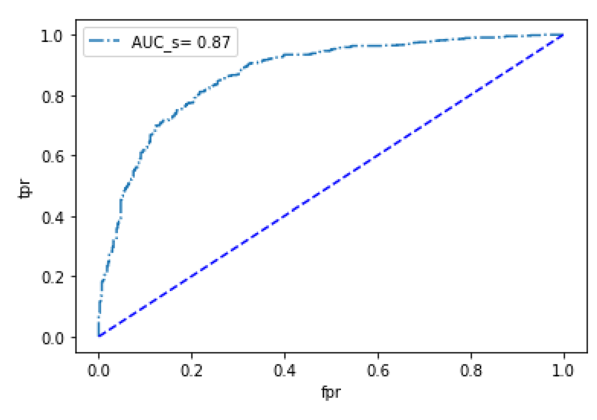
In [43]:
```
# kappa score

metrics.cohen_kappa_score(y_test,y_pred)
```

Out[43]:
```
0.5914724107406315
```

ROC AUC

In [47]:
```
# roc for female
AUC_score = metrics.auc(fpr,tpr)

fpr,tpr,thresh = metrics.roc_curve(y_test,y_prob[:,1])
plt.plot(fpr,tpr,'-.')
plt.plot([0,1],[0,1],'b--')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.legend(['AUC_s= %0.2f'%AUC_score])
```

Out[47]:
```
<matplotlib.legend.Legend at 0x245c3b96b08>
```



Hyperparameter tuning

In [48]:
```
metrics.SCORERS
```

Out[48]:
```
{'explained_variance': make_scorer(explained_variance_score),
 'r2': make_scorer(r2_score),
 'max_error': make_scorer(max_error, greater_is_better=False),
 'neg_median_absolute_error': make_scorer(median_absolute_error, greater_is_better=False),
 'neg_mean_absolute_error': make_scorer(mean_absolute_error, greater_is_better=False),
 'neg_mean_squared_error': make_scorer(mean_squared_error, greater_is_better=False),
 'neg_mean_squared_log_error': make_scorer(mean_squared_log_error, greater_is_better=False),
 'neg_root_mean_squared_error': make_scorer(mean_squared_error, greater_is_better=False, squa
red=False),
 'neg_mean_poisson_deviance': make_scorer(mean_poisson_deviance, greater_is_better=False),
 'neg_mean_gamma_deviance': make_scorer(mean_gamma_deviance, greater_is_better=False),
 'accuracy': make_scorer(accuracy_score),
 'roc_auc': make_scorer(roc_auc_score, needs_threshold=True),
 'roc_auc_ovr': make_scorer(roc_auc_score, needs_proba=True, multi_class=ovr),
 'roc_auc_ovo': make_scorer(roc_auc_score, needs_proba=True, multi_class=ovo),
 'roc_auc_ovr_weighted': make_scorer(roc_auc_score, needs_proba=True, multi_class=ovr, averag
e=weighted),
 'roc_auc_ovo_weighted': make_scorer(roc_auc_score, needs_proba=True, multi_class=ovo, averag
e=weighted),
 'balanced_accuracy': make_scorer(balanced_accuracy_score),
 'average_precision': make_scorer(average_precision_score, needs_threshold=True),
 'neg_log_loss': make_scorer(log_loss, greater_is_better=False, needs_proba=True),
 'neg_brier_score': make_scorer(brier_score_loss, greater_is_better=False, needs_proba=True),
 'adjusted_rand_score': make_scorer(adjusted_rand_score),
 'homogeneity_score': make_scorer(homogeneity_score),
 'completeness_score': make_scorer(completeness_score),
 'v_measure_score': make_scorer(v_measure_score),
 'mutual_info_score': make_scorer(mutual_info_score),
 'adjusted_mutual_info_score': make_scorer(adjusted_mutual_info_score),
 'normalized_mutual_info_score': make_scorer(normalized_mutual_info_score),
 'fowlkes_mallows_score': make_scorer(fowlkes_mallows_score),
 'precision': make_scorer(precision_score, average=binary),
 'precision_macro': make_scorer(precision_score, pos_label=None, average=macro),
 'precision_micro': make_scorer(precision_score, pos_label=None, average=micro),
 'precision_samples': make_scorer(precision_score, pos_label=None, average=samples),
 'precision_weighted': make_scorer(precision_score, pos_label=None, average=weighted),
 'recall': make_scorer(recall_score, average=binary),
 'recall_macro': make_scorer(recall_score, pos_label=None, average=macro),
 'recall_micro': make_scorer(recall_score, pos_label=None, average=micro),
 'recall_samples': make_scorer(recall_score, pos_label=None, average=samples),
 'recall_weighted': make_scorer(recall_score, pos_label=None, average=weighted),
 'f1': make_scorer(f1_score, average=binary),
 'f1_macro': make_scorer(f1_score, pos_label=None, average=macro),
 'f1_micro': make_scorer(f1_score, pos_label=None, average=micro),
 'f1_samples': make_scorer(f1_score, pos_label=None, average=samples),
 'f1_weighted': make_scorer(f1_score, pos_label=None, average=weighted),
 'jaccard': make_scorer(jaccard_score, average=binary),
 'jaccard_macro': make_scorer(jaccard_score, pos_label=None, average=macro),
 'jaccard_micro': make_scorer(jaccard_score, pos_label=None, average=micro),
 'jaccard_samples': make_scorer(jaccard_score, pos_label=None, average=samples),
 'jaccard_weighted': make_scorer(jaccard_score, pos_label=None, average=weighted)}
```

In [49]:
```
model_tune = SVC()
```

In [50]:
```
from sklearn.model_selection import GridSearchCV
```

In [52]:
```
param_grid = { 'C':[1,10,20,30,50,100],
              'kernel':['rbf','poly'],
              'gamma':[0.1,0.005,0.01,0.001,0.002,0.005],
              'coef0':[0,1],}
```

In [54]:
```
model_grid = GridSearchCV(model_tune,param_grid=param_grid,scoring='accuracy',cv=5,verbose=1
)
```

In [55]:
```
model_grid.fit(X,y)
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 720 out of 720 | elapsed: 27.8min finished
```

Out[55]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [1, 10, 20, 30, 50, 100], 'coef0': [0, 1],
                         'gamma': [0.1, 0.005, 0.01, 0.001, 0.002, 0.005],
                         'kernel': ['rbf', 'poly']},
             scoring='accuracy', verbose=1)
```

In [56]:
```
model_grid.best_params_
```

Out[56]:
```
{'C': 10, 'coef0': 0, 'gamma': 0.005, 'kernel': 'rbf'}
```

In [57]:
```
model_grid.best_score_
```

Out[57]:
```
0.7812382698267208
```

In [63]:
```
# Building best ML model with best parameters

model_best = SVC(C=10,kernel='rbf',gamma=0.005,probability=True)
```

In [64]:
```
model_best.fit(x_train,y_train)
```

Out[64]:
```
SVC(C=10, gamma=0.005, probability=True)
```

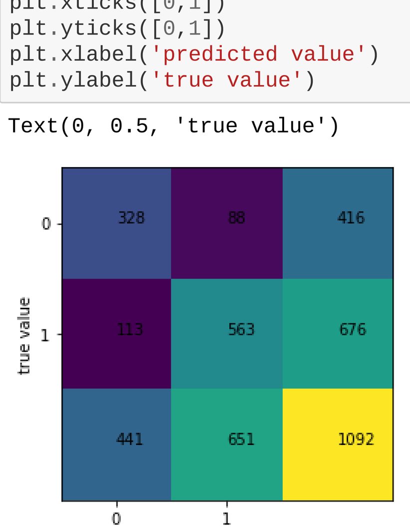
In [65]:
```
model_best.score(x_test,y_test)
```

Out[65]:
```
0.8159340659340659
```

In [66]:
```
y_pred = model_best.predict(x_test)
y_prob = model_best.predict_proba(x_test)
```

In [67]:
```
cm = metrics.confusion_matrix(y_test,y_pred)

cm = np.concatenate((cm,cm.sum(axis=0).reshape(1,-1)),axis=0)
cm = np.concatenate((cm,cm.sum(axis=1).reshape(-1,1)),axis=1)
cm

plt.imshow(cm)
for i in range(3):
    for j in range(3):
        plt.text(i,j,'%d'%cm[i,j])

plt.xticks([0,1])
plt.yticks([0,1])
plt.xlabel('predicted value')
plt.ylabel('true value')
```

Out[67]:
```
Text(0, 0.5, 'true value')
```



In [68]:
```
cr = metrics.classification_report(y_test,y_pred,target_names=['male','female'],output_dict=
True)
pd.DataFrame(cr).T
```

Out[68]:

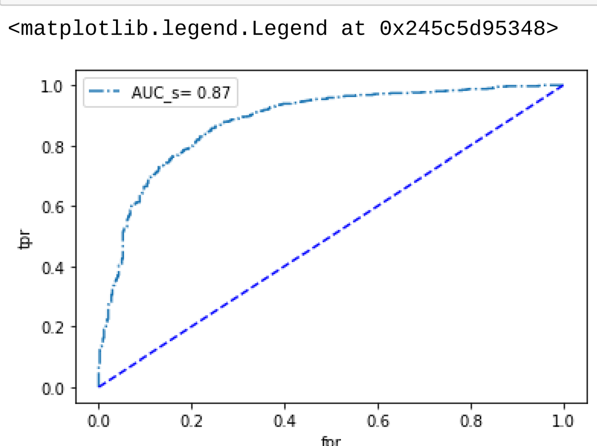
	precision	recall	f1-score	support
male	0.788462	0.743764	0.765461	441.000000
female	0.832840	0.864823	0.848531	651.000000
accuracy	0.815934	0.815934	0.815934	0.815934
macro avg	0.810651	0.804294	0.806996	1092.000000
weighted avg	0.814918	0.815934	0.814983	1092.000000

In [69]:
```
# kappa score

metrics.cohen_kappa_score(y_test,y_pred)
```

Out[69]:
```
0.6142034548944337
```

In [70]:
```
# roc for female
AUC_score = metrics.auc(fpr,tpr)

fpr,tpr,thresh = metrics.roc_curve(y_test,y_prob[:,1])
plt.plot(fpr,tpr,'-.')
plt.plot([0,1],[0,1],'b--')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.legend(['AUC_s= %0.2f'%AUC_score])
```

Out[70]:
```
<matplotlib.legend.Legend at 0x245c5d95348>
```



In [71]:
```
# saving the machine learniing model

import pickle
```

In [72]:
```
pickle.dump(model_best,open('model_cvm.pickle','wb'))
```