In [ ]:
```
#load the file
```

In [1]:
```
from scipy import io as sio
file = sio.loadmat('./EMNIST./zip/matlab/emnist-byclass.mat')
```

In [ ]:
```
#load the data, seperate the data to train set and data set, then reshape
```

In [2]:
```
import numpy as np

print(file.keys())
train = file['dataset'][0][0][0]
test = file['dataset'][0][0][1]
data_train = train[0][0][0]
label_train = train[0][0][1].reshape(-1)
data_test = test[0][0][0]
label_test = test[0][0][1].reshape(-1)

mapping = file['dataset'][0][0][2]

print(data_train.shape)
print(label_train.shape)
print(data_test.shape)
print(label_test.shape)
```

```
dict_keys(['__header__', '__version__', '__globals__', 'dataset'])
(697932, 784)
(697932,)
(116323, 784)
(116323,)
```

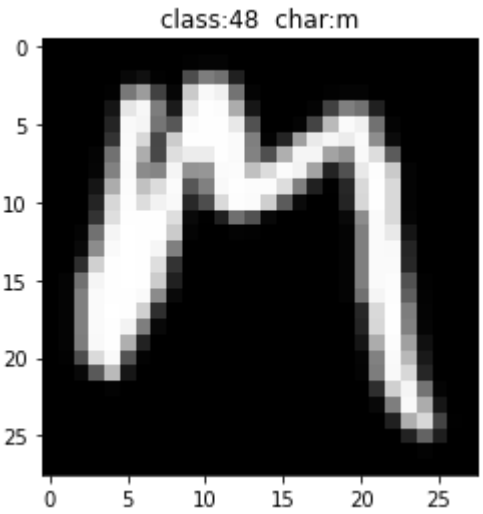
In [ ]:
```
#rotate the image by -90 degree and then flip horizontally
```

In [3]:
```
data_train = data_train.reshape([-1,28,28])
data_train = np.rot90(data_train, axes=[2,1])
data_train = np.flip(data_train, axis=2)
data_train = data_train.reshape([-1, 28 * 28])
data_test = data_test.reshape([-1,28,28])
data_test = np.rot90(data_test, axes=[2,1])
data_test = np.flip(data_test, axis=2)
data_test= data_test.reshape([-1, 28 * 28])

#randomly shuffle the data, and take 10% of the dataset
#1/10 for train and test respectively, 69793 train samples and 11632 test sam

train_sample_index = np.random.choice(data_train.shape[0], data_train.shape[0
data_train = data_train[train_sample_index]
label_train = label_train[train_sample_index]

test_sample_index = np.random.choice(data_test.shape[0], data_test.shape[0]//
data_test = data_test[test_sample_index]
label_test = label_test[test_sample_index]

print(data_train.shape)
print(label_train.shape)
print(data_test.shape)
print(label_test.shape)
```

```
(69793, 784)
(69793,)
(11632, 784)
(11632,)
```

In [ ]:
```
#show the original image in dataset
```

In [4]:
```
import matplotlib.pyplot as plt
img = data_train[0].reshape([28,28])
plt.imshow(img, cmap=plt.get_cmap('gray'))
plt.title("class:" +  str(label_train[0]) + "  char:" + chr(mapping[label_tra
plt.show()
```



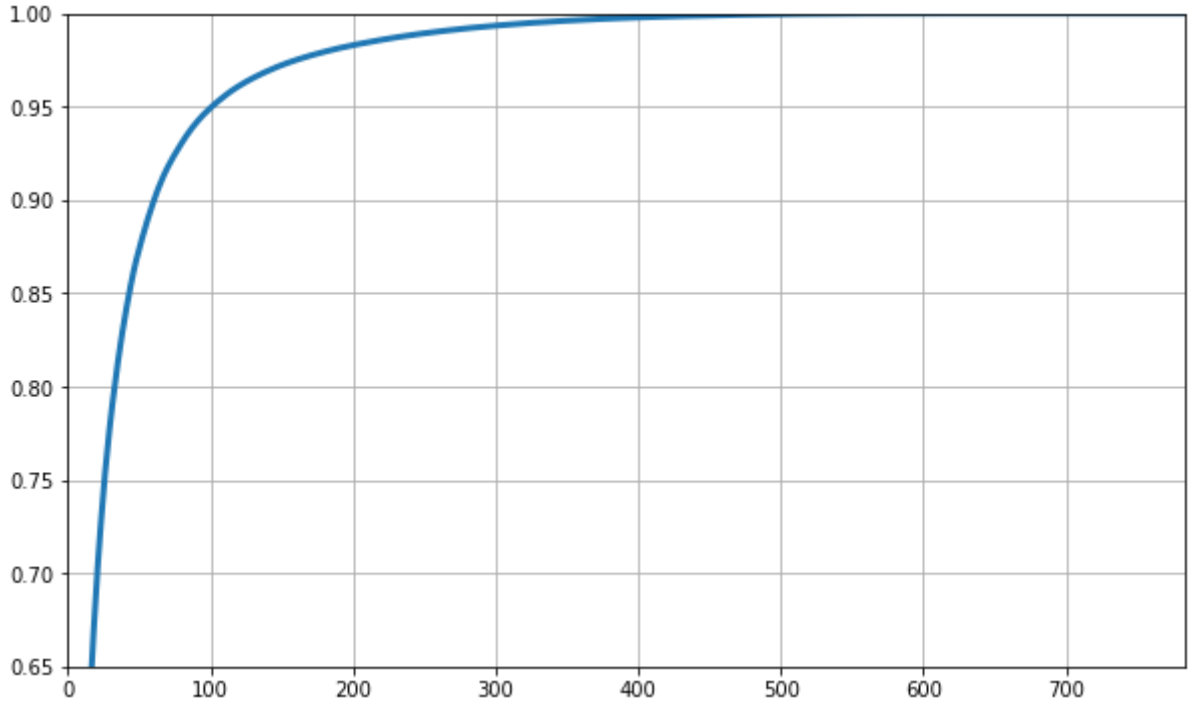
In [ ]:
```

```

In [5]:
```
# Pre-Processing Normaliazation
```

In [6]:
```
# all feature data are divided by 255 to scale all image pixels to 0-1.
data_train = data_train/255
data_test = data_test/255
```

In [7]:
```
# Pre-Processing PCA
```

In [8]:
```
from sklearn.decomposition import PCA

pca = PCA()
#train
pca.fit(data_train)

cumsum = np.cumsum(pca.explained_variance_ratio_)

#show the image of the variance occupied by each dimension
plt.figure(figsize = (10,6) )
plt.plot(cumsum, linewidth = 3)
plt.axis([0,784,0.65,1])
plt.grid(True)
plt.show()
```



In [9]:
```
# reduce the dimension 150, because when the dimension is 150, more than 95% (
pca = PCA(n_components=150)
pca.fit(data_train)


data_train = pca.transform(data_train)

data_test = pca.transform(data_test)
```

In [10]:
```
data_train.shape
```

Out[10]: (69793, 150)

In [ ]:
```

```

In [11]:
```
# SVM classifier
```

In [ ]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

#set the parameters
grid_params = {'C': [0.001, 0.005, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07,

#train the model
svm = GridSearchCV(SVC(), grid_params, cv = 5, n_jobs = -1, verbose=10)
svm.fit(data_train, label_train)
```

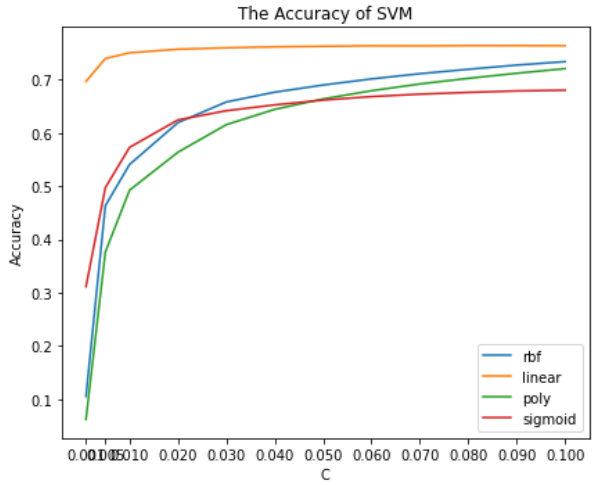
```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```

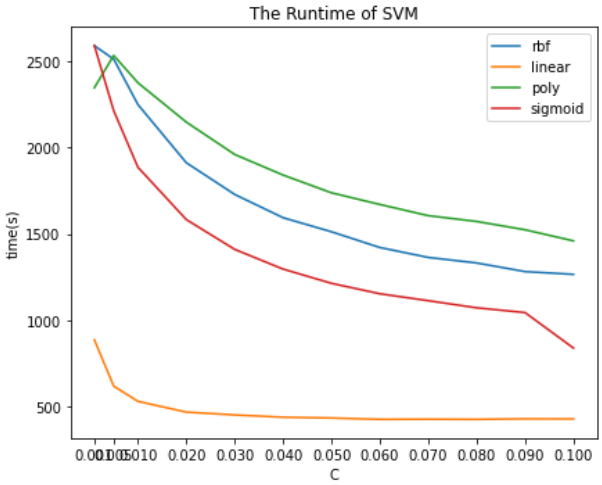
In [12]:
```
#show the best search score
best_cv_score = np.max(svm.cv_results_['mean_test_score'])
best_params = svm.best_params_
print(f'best cv score {best_cv_score}')
```

In [ ]:
```
# Graphs of the accuracy vs. c, the runtime vs. c
# Graphs of the accuracy vs. kernel, the runtime vs. kernel
```

In [52]:
```
cv_results = svm.cv_results_
cost_times = cv_results['mean_fit_time'] + cv_results['mean_score_time']
kernal_cost_time = []
kernal_score = []
for kernal in grid_params['kernel']:
    index = cv_results['param_kernel'] == kernal
    score = list(cv_results['mean_test_score'][index])
    cost_time = list(cost_times[index])
    kernal_score.append(score)
    kernal_cost_time.append(cost_time)

plt.figure(figsize = (15,12))

plt.subplot(2,2,1)
for score in kernal_score:
    print(score)
    plt.plot(grid_params['C'], score)
plt.legend(grid_params['kernel'])
plt.xticks(grid_params['C'])
plt.title("The Accuracy of SVM")
plt.ylabel("Accuracy")
plt.xlabel("C")

plt.subplot(2,2,2)
for cost_time in kernal_cost_time:
    plt.plot(grid_params['C'], cost_time)
plt.legend(grid_params['kernel'])
plt.xticks(grid_params['C'])
plt.title("The Runtime of SVM")
plt.ylabel("time(s)")
plt.xlabel("C")

plt.show()
```

```
[0.10528276159211518, 0.4633702536282617, 0.5406415525474831, 0.61927408929478
98, 0.6580745747522674, 0.6764574812932652, 0.6898398992788544, 0.701345343738
4807, 0.7111600703269327, 0.7194273475713541, 0.727350799648544, 0.73378410897
13729]
[0.6966170594310332, 0.739586983192267, 0.7503617193623382, 0.757110290888220
3, 0.7598039962302965, 0.7615950006333924, 0.7626695967057477, 0.7635722565853
383, 0.7634003378428142, 0.7638874838879107, 0.7639304679361019, 0.76367255543
50771]
[0.06206925608321284, 0.3762842188822257, 0.4923702919264071, 0.56372414963720
8, 0.6154771469033403, 0.644205005589158, 0.6638057982858342, 0.67916549583251
82, 0.6919031561746942, 0.7024199428902902, 0.7119481447422717, 0.720645300448
54]
[0.3113206720752763, 0.4972275170870958, 0.5727938197325082, 0.624346275755823
6, 0.6416115990968208, 0.6526012236868208, 0.6615849380035556, 0.6681758527657
928, 0.6727321577681168, 0.6760419469290764, 0.6788072598461212, 0.68012544382
50532]
```





In [ ]:
```
#calculate the best parameters of SVM model
```

In [13]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, co

best_esemble = SVC(**best_params)
best_esemble.fit(data_train, label_train)
```

Out[13]:
```
SVC(C=0.09, kernel='linear')
```

In [ ]:
```
#calculate the perfermance metric of SVM model, including precision, recall,
```

In [14]:
```
predict_data = best_esemble.predict(data_test)
precision = precision_score(label_test, predict_data, average = 'macro')
recall = recall_score(label_test, predict_data, average = 'macro')
accuracy = accuracy_score(label_test, predict_data)

print("Accuracy" + str(accuracy))
print("Recall" + str(recall))
print("Precision" + str(precision))

#show the confusion_matrix of this SVM model
print(confusion_matrix(predict_data,label_test))
```

```
Accuracy0.7575653370013755
Recall0.5733782322120649
Precision0.6203616955697413
[[436   0   1 ...   0   0   0]
 [  0 611   1 ...   1   0   0]
 [  1   0 542 ...   0   0  18]
 ...
 [  0   0   0 ...  25   0   0]
 [  0   1   0 ...   1   6   0]
 [  0   0   3 ...   0   0  13]]
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/
sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
o_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [55]:
```
#save the model
```

In [56]:
```
import joblib
joblib.dump(best_esemble,'./EMNIST./zip/matlab/svm.model')
best_esemble = joblib.load('./EMNIST./zip/matlab/svm.model')
```

In [ ]:
```

```