Carlos Jaime
1000847444
INSY5375 - Data Science: A programming Approach

12.9.2019

Individual Assignment

In this project, i used the following Supervised models for image classification:

1. Logistic
2. SVM
3. Nerual Neworks
4. KNN
5. Naive Bayes

All 5 of my images were taken with an iphone 11 pro.
The images were then opened in photoshop and the backgrounds were removed using the magic wand tool.
I then applied a white background to all 5 images (more information below).

Fashion_mnist data:

In [1]:
```
#Here, i made my data ready before i applied different models.

import tensorflow as tf
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist

import numpy as np
```

In [2]:
```
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

In [3]:
```
#changed range of the data
X_train = X_train / 255.0
X_test = X_test / 255.0
```

In [4]:

```
#reshaping train data into 1 dimension, flat the images
X_train_flat =  []


for x in X_train:
    X_train_flat.append(x.reshape(28*28))


X_train_flat= np.array(X_train_flat)
X_train_flat.shape
```

Out[4]:

```
(60000, 784)
```

In [5]:

```
#reshaping test data into 1 dimension
X_test_flat =  []



for x in X_test:
    X_test_flat.append(x.reshape(28*28))


X_test_flat= np.array(X_test_flat)
X_test_flat.shape
```

Out[5]:

```
(10000, 784)
```

In [6]:

```
print(X_train_flat.shape)
print(y_train.shape)
print('----')
print(X_test_flat.shape)
print(y_test.shape)
```

```
(60000, 784)
(60000,)
----
(10000, 784)
(10000,)
```

In [7]:

```
#reshaped the target variables
y_train.reshape(60000,1)
y_test.reshape(10000,1)
```

Out[7]:

```
array([[9],
       [2],
       [1],
       ...,
       [8],
       [1],
       [5]], dtype=uint8)
```

Task 1

Model 1: Logistic

In [8]:

```
X_test_flat_logistic = X_test_flat.copy()
X_train_flat_logistic = X_train_flat.copy()

y_test_logistic = y_test.copy()
y_train_logistic = y_train.copy()
```

In [9]:

```
# #model
# # Step 1
from sklearn.linear_model import LogisticRegression
# # Step 2
model_log = LogisticRegression(max_iter=10000)
# # Step 3
# #X_train, X_test, y_train, y_test = train_test_split(X_train_flat, y_train,
# # Step 4
model_log.fit(X_train_flat_logistic, y_train_logistic)
# # Step 5
y_test_hat_logistic = model_log.predict(X_test_flat_logistic)
y_train_hat_logistic = model_log.predict(X_train_flat_logistic)
```

In [10]:

```
from sklearn.metrics import accuracy_score
print('in sample:{}'.format(accuracy_score(y_train_logistic,y_train_hat_logis
print('out sample:{}'.format(accuracy_score(y_test_logistic,y_test_hat_logist
```

```
in sample:88.10833333333333
out sample:84.41
```

In [11]:
```
class_names = ['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Sh

from sklearn.metrics import classification_report
print(classification_report(y_test_logistic, y_test_hat_logistic,
                            target_names=class_names))
```

```
              precision    recall  f1-score   support

 T-shirt/top       0.80      0.81      0.80      1000
     Trouser       0.97      0.96      0.96      1000
    Pullover       0.73      0.74      0.73      1000
       Dress       0.83      0.86      0.84      1000
        Coat       0.74      0.76      0.75      1000
      Sandal       0.94      0.92      0.93      1000
       Shirt       0.63      0.57      0.60      1000
     Sneaker       0.91      0.94      0.92      1000
         Bag       0.93      0.93      0.93      1000
  Ankle boot       0.95      0.95      0.95      1000

    accuracy                           0.84     10000
   macro avg       0.84      0.84      0.84     10000
weighted avg       0.84      0.84      0.84     10000
```

In [13]:
```
from sklearn.metrics import roc_auc_score

log_auc = roc_auc_score(y_test_logistic, model_log.predict_proba(X_test_flat_
print("AUC for log: {:.3f}".format(log_auc))
```

```
AUC for log: 0.983
```

Model 2: SVM

In [14]:
```
X_test_flat_svm = X_test_flat.copy()
X_train_flat_svm = X_train_flat.copy()

y_test_svm = y_test.copy()
y_train_svm = y_train.copy()
```

In [15]:

```
#here i created a pipeline to handle the svc model and pca reduction
#i recuded the dimension to 150


from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC




from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
# InitializeDimension Reduction model
pca = PCA(svd_solver='randomized', n_components=150,
          whiten=True, random_state=0)

svc = SVC(probability=True)
# Create pipleline model
model_svm = make_pipeline(pca, svc)
```

In [16]:

```
model_svm.fit(X_train_flat_svm, y_train_svm)
```

Out[16]:

```
Pipeline(steps=[('pca',
                 PCA(n_components=150, random_state=0, svd_solver='randomiz
ed',
                     whiten=True)),
                ('svc', SVC(probability=True))])
```

In [17]:
```python
class_names = ['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Sh

y_test_hat_svm = model_svm.predict(X_test_flat_svm)
y_train_hat_svm = model_svm.predict(X_train_flat_svm)

from sklearn.metrics import classification_report
print(classification_report(y_test_svm, y_test_hat_svm,
                            target_names=class_names))
```

```
              precision    recall  f1-score   support

 T-shirt/top       0.84      0.86      0.85      1000
     Trouser       1.00      0.97      0.98      1000
    Pullover       0.81      0.82      0.81      1000
       Dress       0.89      0.90      0.90      1000
        Coat       0.82      0.82      0.82      1000
      Sandal       0.96      0.98      0.97      1000
       Shirt       0.73      0.68      0.71      1000
     Sneaker       0.95      0.97      0.96      1000
         Bag       0.97      0.98      0.97      1000
  Ankle boot       0.98      0.96      0.97      1000

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

In [18]:
```python
#very good variance and bias
from sklearn.metrics import accuracy_score
print('out sample:{}'.format(accuracy_score(y_train_svm,y_train_hat_svm, norm
print('out sample:{}'.format(accuracy_score(y_test_svm,y_test_hat_svm, normal
```

```
out sample:95.42333333333333
out sample:89.46
```

In [19]:
```python
#very good AUC
from sklearn.metrics import roc_auc_score

log_auc = roc_auc_score(y_test, model_svm.predict_proba(X_test_flat_svm)[:],
print("AUC for log: {:.3f}".format(log_auc))
```

```
AUC for log: 0.992
```

Model 3: Neural Network

In [20]:
```python
X_test_flat_NN =  X_test_flat.copy()
X_train_flat_NN =  X_train_flat.copy()

y_test_NN = y_test.copy()
y_train_NN = y_train.copy()
```

In [21]:

```
#featured reduced to 20
from sklearn.decomposition import PCA
pca_1NN = PCA(svd_solver='randomized',n_components=20)
X_test_flat_NN = pca_1NN.fit_transform(X_test_flat_NN)     # Fit the PCA mode
X_test_flat_NN.shape

from sklearn.decomposition import PCA
pca_2NN = PCA(svd_solver='randomized',n_components=20)
X_train_flat_NN = pca_2NN.fit_transform(X_train_flat_NN)     # Fit the PCA mo
X_train_flat_NN.shape
```

Out[21]:

```
(60000, 20)
```

In [22]:

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score



model_NN = MLPClassifier(solver='lbfgs',random_state=0,
                      hidden_layer_sizes=[10,5])
model_NN.fit(X_train_flat_NN,y_train_NN)

# check the accuracy
y_train_hat_NN = model_NN.predict(X_train_flat_NN)
y_test_hat_NN = model_NN.predict(X_test_flat_NN)

in_sample_acc = accuracy_score(y_train,y_train_hat_NN, normalize = True) * 10
out_of_sample_acc = accuracy_score(y_test,y_test_hat_NN, normalize = True) *
print("In-sample Accuracy: ", in_sample_acc)
print("Out-of-sample Accuracy: ", out_of_sample_acc)
#Here, there seems to be a very serious problem with variance.
#I think its because neural netowrk is too complicated and therefore we are g
```

```
In-sample Accuracy:  82.75666666666666
Out-of-sample Accuracy:  47.19

C:\Users\Carlos\Anaconda3\lib\site-packages\sklearn\neural_network\_multila
yer_perceptron.py:471: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sci
kit-learn.org/stable/modules/preprocessing.html)
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

In [23]:
```
class_names = ['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Sh


from sklearn.metrics import classification_report
print(classification_report(y_test_NN, y_test_hat_NN,
                            target_names=class_names))
```

```
              precision    recall  f1-score   support

 T-shirt/top       0.67      0.61      0.64      1000
     Trouser       0.86      0.90      0.88      1000
    Pullover       0.49      0.24      0.32      1000
       Dress       0.52      0.62      0.57      1000
        Coat       0.54      0.32      0.40      1000
      Sandal       0.35      0.38      0.37      1000
       Shirt       0.15      0.18      0.17      1000
     Sneaker       0.43      0.68      0.53      1000
         Bag       0.58      0.79      0.67      1000
  Ankle boot       0.00      0.00      0.00      1000

    accuracy                           0.47     10000
   macro avg       0.46      0.47      0.45     10000
weighted avg       0.46      0.47      0.45     10000
```

In [24]:
```
#AUC score is ok
from sklearn.metrics import roc_auc_score

log_auc = roc_auc_score(y_test, model_NN.predict_proba(X_test_flat_NN)[:], mu
print("AUC for log: {:.3f}".format(log_auc))
```

```
AUC for log: 0.787
```

Model 4: KNN

In [25]:
```
X_test_flat_KNN = X_test_flat.copy()
X_train_flat_KNN = X_train_flat.copy()

y_test_KNN = y_test.copy()
y_train_KNN = y_train.copy()
```

In [26]:
```
X_test_flat_KNN.shape
```

Out[26]:
```
(10000, 784)
```

In [27]:

```python
from sklearn.decomposition import PCA
pca_KNN1 = PCA(svd_solver='randomized',n_components=40)
X_test_flat_KNN = pca_KNN1.fit_transform(X_test_flat_KNN)     # Fit the PCA m
X_test_flat_KNN.shape

pca_KNN2 = PCA(svd_solver='randomized', n_components=40)
X_train_flat_KNN = pca_KNN2.fit_transform(X_train_flat_KNN)     # Fit the PCA
X_train_flat_KNN.shape
```

Out[27]:

```
(60000, 40)
```

In [28]:

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
```

In [29]:

```python
model_KNN = KNeighborsClassifier(n_neighbors=7)
model_KNN.fit(X_train_flat_KNN, y_train_KNN);

from sklearn.metrics import classification_report, confusion_matrix

y_train_hat_KNN = model_KNN.predict(X_train_flat_KNN)
y_test_hat_KNN = model_KNN.predict(X_test_flat_KNN)
```

In [30]:

```python
#There seems to be a lot if variance here.
print('in sample:{}'.format(accuracy_score(y_train,y_train_hat_KNN, normalize
print('out sample:{}'.format(accuracy_score(y_test,y_test_hat_KNN, normalize
```

```
in sample:89.12166666666667
out sample:56.21000000000001
```

In [31]:

```
class_names = ['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Sh


from sklearn.metrics import classification_report
print(classification_report(y_test_KNN, y_test_hat_KNN,
                            target_names=class_names))
```

```
              precision    recall  f1-score   support

 T-shirt/top       0.65      0.73      0.69      1000
     Trouser       0.95      0.91      0.93      1000
    Pullover       0.39      0.37      0.38      1000
       Dress       0.62      0.72      0.67      1000
        Coat       0.47      0.57      0.52      1000
      Sandal       0.43      0.40      0.42      1000
       Shirt       0.28      0.17      0.21      1000
     Sneaker       0.59      0.82      0.69      1000
         Bag       0.80      0.91      0.85      1000
  Ankle boot       0.03      0.01      0.02      1000

    accuracy                           0.56     10000
   macro avg       0.52      0.56      0.54     10000
weighted avg       0.52      0.56      0.54     10000
```

In [1]:

```
#very good auc
from sklearn.metrics import roc_auc_score

log_auc = roc_auc_score(y_test, model_KNN.predict_proba(X_test_flat_KNN)[:],
print("AUC for log: {:.3f}".format(log_auc))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-03dfd375fa21> in <module>
      2 from sklearn.metrics import roc_auc_score
      3
----> 4 log_auc = roc_auc_score(y_test, model_KNN.predict_proba(X_test_flat
_KNN)[:], multi_class="ovr")
      5 print("AUC for log: {:.3f}".format(log_auc))

NameError: name 'y_test' is not defined
```

Model 5: Naive Bayes

In [141]:

```
X_test_flat_NB = X_test_flat.copy()
X_train_flat_NB = X_train_flat.copy()

y_test_NB = y_test.copy()
y_train_NB = y_train.copy()
```

In [142]:

```
#i reduced the amount of features to 140

pca_NB1 = PCA(svd_solver='randomized', n_components=140)
X_train_flat_NB = pca_NB1.fit_transform(X_train_flat_NB)     # Fit the PCA mo
X_train_flat_NB.shape

# from sklearn.manifold import Isomap
# iso = Isomap(n_components=45)
# X_train_flat = iso.fit_transform(X_train_flat)
```

Out[142]:

```
(60000, 140)
```

In [143]:

```
#i reduced the amount of features to 140

pca_NB2 = PCA(svd_solver='randomized', n_components=140)
X_test_flat_NB = pca_NB2.fit_transform(X_test_flat_NB)     # Fit the PCA mode
X_test_flat_NB.shape

# from sklearn.manifold import Isomap
# iso = Isomap(n_components=45)
# X_train_flat = iso.fit_transform(X_train_flat)
```

Out[143]:

```
(10000, 140)
```

In [144]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
model_NB = GaussianNB()
model_NB.fit(X_train_flat_NB, y_train_NB)



y_train_hat_NB = model_NB.predict(X_train_flat_NB)
y_test_hat_NB = model_NB.predict(X_test_flat_NB)
```

In [145]:

```
#here seems a lot of variance and bias here

from sklearn.metrics import accuracy_score
print('in sample:{}'.format(accuracy_score(y_train_NB,y_train_hat_NB, normali
print('out sample:{}'.format(accuracy_score(y_test_NB,y_test_hat_NB, normaliz
```

```
in sample:76.31166666666667
out sample:52.580000000000005
```

In [146]:

```
class_names = ['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Sh


from sklearn.metrics import classification_report
print(classification_report(y_test_NB, y_test_hat_NB,
                            target_names=class_names))
```

```
              precision    recall  f1-score   support

 T-shirt/top       0.46      0.73      0.57      1000
     Trouser       0.97      0.72      0.83      1000
    Pullover       0.27      0.16      0.20      1000
       Dress       0.74      0.56      0.64      1000
        Coat       0.48      0.58      0.53      1000
      Sandal       0.59      0.64      0.62      1000
       Shirt       0.23      0.26      0.24      1000
     Sneaker       0.80      0.69      0.74      1000
         Bag       0.46      0.84      0.59      1000
  Ankle boot       0.31      0.07      0.12      1000

    accuracy                           0.53     10000
   macro avg       0.53      0.53      0.51     10000
weighted avg       0.53      0.53      0.51     10000
```

In [147]:

```
#auc score is very good but accuracy, precison and f1 is not very good
from sklearn.metrics import roc_auc_score

log_auc = roc_auc_score(y_test, model_NB.predict_proba(X_test_flat_NB)[:], mu
print("AUC for log: {:.3f}".format(log_auc))
```

```
AUC for log: 0.876
```

In [ ]:

In [ ]:

In [ ]:

Task 2

In [99]:

```
#Here, im opening each of my images and coverting them to a Grayscale format,
#I then add all the images to X_experiment, so i can loop through the images
import cv2
import numpy as np
X_experiment =  []




#image 1#
img = cv2.imread('shoes_clean.jpg')
res = cv2.resize(img, dsize=(28, 28), interpolation=cv2.INTER_CUBIC)
res = 255-res
img=res / 255

img = np.mean(img, axis=2)
X_experiment.append(img.reshape(28*28))





#image 2#
img = cv2.imread('shirt_1_clean.jpg')
res = cv2.resize(img, dsize=(28, 28), interpolation=cv2.INTER_CUBIC)
res = 255-res
img=res / 255

img = np.mean(img, axis=2)
X_experiment.append(img.reshape(28*28))
#     #print(x.reshape(28*28))






#image 3#
img = cv2.imread('shirt_2_clean.jpg')
res = cv2.resize(img, dsize=(28, 28), interpolation=cv2.INTER_CUBIC)
res = 255-res
img=res / 255

img = np.mean(img, axis=2)
X_experiment.append(img.reshape(28*28))



#image 4#
img = cv2.imread('trouser_clean.jpg')
res = cv2.resize(img, dsize=(28, 28), interpolation=cv2.INTER_CUBIC)
res = 255-res
img=res / 255

img = np.mean(img, axis=2)
```

```
X_experiment.append(img.reshape(28*28))





#image 5#
img = cv2.imread('pullover_clean.jpg')
res = cv2.resize(img, dsize=(28, 28), interpolation=cv2.INTER_CUBIC)
res = 255-res
img=res / 255

img = np.mean(img, axis=2)
#X_experiment.append(img.reshape(28*28))
X_experiment.append(img.reshape(28*28))





X_experiment= np.array(X_experiment)
X_experiment.shape
```

Out[99]:

```
(5, 784)
```

Pictures of my five fashion clothing items i took

In [138]:

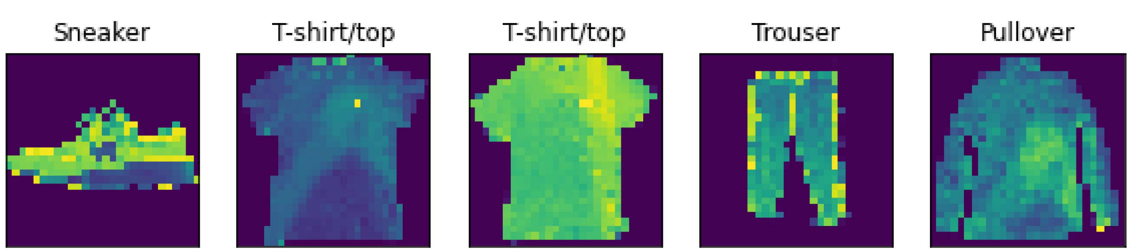
```
#here i create a list of the images names and then open all the images
#these are the orginal pictures i took, raw images

import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
%matplotlib inline
img=[]
filess = ['shoes','shirt1','shirt2','trouser','pullover']
for x in filess:
    #print(x+'.jpg')
    fashion_image = np.array(mpimg.imread('./original_images/'+x+'.jpg'))
    img.append(fashion_image)



#my images i took
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 5, figsize=(10, 10))
for i, axi in enumerate(ax.flat):
    axi.imshow(img[i])
    #axi.set_title(class_names[my_label[i]])
    axi.set(xticks=[], yticks=[])
```

In [135]:

```
#the previous images were then opened in photoshop and the bacgkround was rem
#here i view the new images exported from photoshop with a new white bacgrkou
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
%matplotlib inline
img=[]
filess = ['shoes_clean','shirt_1_clean','shirt_2_clean','trouser_clean','pull
for x in filess:
    #print(x+'.jpg')
    fashion_image = np.array(mpimg.imread(x+'.jpg'))
    img.append(fashion_image)




#my images i took
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 5, figsize=(10, 10))
for i, axi in enumerate(ax.flat):
    axi.imshow(img[i])
    #axi.set_title(class_names[my_label[i]])
    axi.set(xticks=[], yticks=[])
```

In [140]:

```
#here, i loop through the images and view them in the same manner the fashion
my_label =  [7,0,0,1,2]

import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 5, figsize=(10, 10))
for i, axi in enumerate(ax.flat):
    axi.imshow(X_experiment[i].reshape(28,28))
    axi.set_title(class_names[my_label[i]])
    axi.set(xticks=[], yticks=[])
```



In [102]:
```python
# here i create a list containing all the labels for my images
#example 7, standers for a sneaker, and its in the same position as X_experim
import numpy as np
#the categories my images belong with
#
y_experiment =  [7,0,0,1,2]
#
```

In [103]:
```python
X_experiment.shape
```

Out[103]: (5, 784)

Using Logistic for image classification

In [104]:
```python
#logistic model image classification
y_hat_experiment = model_log.predict(X_experiment)


from sklearn.metrics import accuracy_score
print('score:{}'.format(accuracy_score(y_experiment,y_hat_experiment, normali
```

score:80.0

In [ ]:
```python
#logistic model was able to identify 4 of my 5 images. 80% accuracy
#the image it could not classify corectly was a pullover that looks like a ts
#it was able to classify the pullover as a tshirt.
```

In [149]:
```python
#svm model image classification
y_hat_experiment = model_svm.predict(X_experiment)


from sklearn.metrics import accuracy_score
print('score:{}'.format(accuracy_score(y_experiment,y_hat_experiment, normali
```

score:40.0

In [ ]:

In [ ]:

Extra

I decided to try different ways to make the KNN model better with less train and test data.

Previously, KNN had very high variance and could suffer from overfitting, lots of variables and complexity issues. I decided to draw a validation curve to see how the model perform with different hyperparameters. I then decided to use gridsearch/stratifiedfold to see what it chooses as the best hyperparameters. By comparing these this new model and the old model found above. This new version performed much better.

Previously:
KNN
in sample accuracy: 89%
out of sample accuracy: 52%
This model had 37% variance and 11% bias.
The AUC score was 0.822

new:
KNN
now with the new changes below*

in sample accuracy: 85%
out of sample accuracy: 69%
This model had 15% variance and 16% bias.
The AUC score was 0.945

variance decreased by 22%, bias increased by 4% and AUC incrased by 12%

In [8]:
```
X_test_flat_KNN2 = X_test_flat[:5000].copy()
X_train_flat_KNN2 = X_train_flat[:15000].copy()

y_test_KNN2 = y_test[:5000].copy()
y_train_KNN2 = y_train[:15000].copy()
```

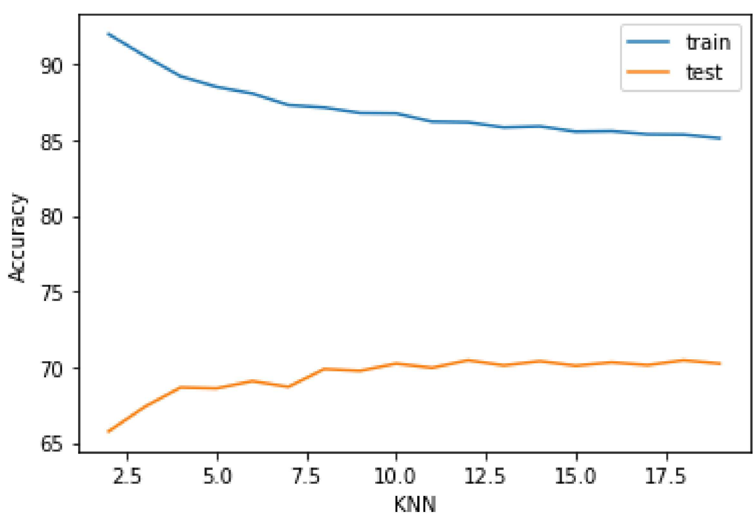
In [9]:
```
from sklearn.decomposition import PCA
pca_KNN1 = PCA(svd_solver='randomized',n_components=40)
X_test_flat_KNN2 = pca_KNN1.fit_transform(X_test_flat_KNN2)     # Fit the PCA
X_test_flat_KNN2.shape

pca_KNN2 = PCA(svd_solver='randomized', n_components=40)
X_train_flat_KNN2 = pca_KNN2.fit_transform(X_train_flat_KNN2)     # Fit the P
X_train_flat_KNN2.shape
```

Out[9]: (15000, 40)

In [10]:

```
#graphing accuracy scores based on hyperparameters neighbors
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_auc_score
n = range(2,20)
score_train = []
score_test = []


for i in n:
    model = model = KNeighborsClassifier(n_neighbors=i).fit(X_train_flat_KNN2
    y_train_hat_KNN2  = model.predict(X_train_flat_KNN2)
    y_test_hat_KNN2  = model.predict(X_test_flat_KNN2)

    score_train.append(accuracy_score(y_train_KNN2,y_train_hat_KNN2, normaliz
    score_test.append(accuracy_score(y_test_KNN2,y_test_hat_KNN2, normalize =


plt.plot(n,  score_train, label='train')
plt.plot(n,  score_test, label='test')
plt.xlabel("KNN")
plt.ylabel("Accuracy")
plt.legend()
```

Out[10]: <matplotlib.legend.Legend at 0x1fea59d8790>



The variance seems to decrease when i use higher knn. The sweet spot seems to be between 7.5 and 12
Next i will run gridsearch to find out the best.

In [11]:
```
KNN_model = KNeighborsClassifier()
param_grid = {'n_neighbors': [5,6,7,8,9,10,15]}
cv = StratifiedKFold(n_splits=5, random_state=0, shuffle=True)
grid = GridSearchCV(KNN_model, param_grid, cv = cv, scoring='accuracy',
                    return_train_score=True)
grid.fit(X_train_flat_KNN2, y_train_KNN2)

print("Best Parameter: {}".format(grid.best_params_))
print("Best Cross Vlidation Score: {}".format(grid.best_score_))
```

```
Best Parameter: {'n_neighbors': 9}
Best Cross Vlidation Score: 0.8374
```

In [12]:
```
bestModel = grid.best_estimator_
y_test_hat_KNN2 = bestModel.predict(X_test_flat_KNN2)
```

In [13]:
```
print('in sample:{}'.format(accuracy_score(y_train_KNN2,y_train_hat_KNN2, nor
print('out sample:{}'.format(accuracy_score(y_test_KNN2,y_test_hat_KNN2, norm
```

```
in sample:85.12666666666667
out sample:69.76
```

In [15]:
```
from sklearn.metrics import roc_auc_score

log_auc = roc_auc_score(y_test_KNN2, model.predict_proba(X_test_flat_KNN2)[:]
print("AUC for log: {:.3f}".format(log_auc))
```

```
AUC for log: 0.945
```

In [ ]: