گزارش – پروژه Ensemble Learning

در بخش اول از این تمرین، باید دیتاست را لود کنیم. چون از محیط Google Colab برای اجرای برنامه استفاده میکنم، از دستور زیر استفاده میکنیم.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

در مرحله بعدی، از فایلهای pairsDevTrain.txt و pairsDevTest.txt که مشخص کننده ی داده های آموزشی و تست هستند، باید عکسهای آموزشی و تست را پیدا کنیم. پس لازم است این دو فایل را بصورت خط به خط بخوانیم. مشخص است که باید علائمی نظیر ‘]’ و ’[’ و ’\t’ و ’\n’ را از خطوط حذف کنیم:

```
lines = []
with open('/content/gdrive/My Drive/payanterm_hoosh/pairsDevTrain.txt', 'r') as f:
    lines = f.readlines()
```

```
lines_test = []
with open('/content/gdrive/My Drive/payanterm_hoosh/pairsDevTest.txt', 'r') as f:
    lines_test = f.readlines()
```

```
lin = []
for line in lines:
  line = line.replace("\n" , "")
  x = line.split("\t")
  lin.append(x)
```

```
lin_test = []
for line in lines_test:
  line = line.replace("\n" , "")
  x = line.split("\t")
  lin_test.append(x)
```

حالا باید داده های آموزشی و تست را از پوشه افراد داده شده بخوانیم. برای این کار ابتدا سراغ داده هایی میرویم که مربوط به یک نفر هستند. با توجه به اسم شخص و شماره ای که جلوی نام او نوشته شده، درون پوشه ها به جستجو میپردازیم.

```
Xtrain_first = []
Xtrain_second = []
Ytrain = []

for i in range(int(lin[0][0])):
  Ytrain.append(1)
  if int(lin[i+1][1]) < 10:
    st = f'000{lin[i+1][1]}'
  elif int(lin[i+1][1]) < 100:
    st = f'00{lin[i+1][1]}'
  elif int(lin[i+1][1]) < 1000:
    st = f'0{lin[i+1][1]}'
  else:
    st = lin[i+1][1]

  if int(lin[i+1][2]) < 10:
    st1 = f'000{lin[i+1][2]}'
  elif int(lin[i+1][2]) < 100:
    st1 = f'00{lin[i+1][2]}'
  elif int(lin[i+1][2]) < 1000:
    st1 = f'0{lin[i+1][2]}'
  else:
    st1 = lin[i+1][2]

  x = open(f'/content/gdrive/My Drive/payanterm_hoosh/lfw/{lin[i+1][0]}/{lin[i+1][0]}_{st}.txt', 'r')
  y = open(f'/content/gdrive/My Drive/payanterm_hoosh/lfw/{lin[i+1][0]}/{lin[i+1][0]}_{st1}.txt', 'r')

```

این کار را برای داده هایی که متعلق به دو نفر متفاوت هستند نیز تکرار میکنیم.

```
for i in range(int(lin[0][0]), 2*int(lin[0][0])):
  Ytrain.append(0)
  if int(lin[i+1][1]) < 10:
    st = f'000{lin[i+1][1]}'
  elif int(lin[i+1][1]) < 100:
    st = f'00{lin[i+1][1]}'
  elif int(lin[i+1][1]) < 1000:
    st = f'0{lin[i+1][1]}'
  else:
    st = lin[i+1][1]

  if int(lin[i+1][3]) < 10:
    st1 = f'000{lin[i+1][3]}'
  elif int(lin[i+1][3]) < 100:
    st1 = f'00{lin[i+1][3]}'
  elif int(lin[i+1][3]) < 1000:
    st1 = f'0{lin[i+1][3]}'
  else:
    st1 = lin[i+1][3]

  x = open(f'/content/gdrive/My Drive/payanterm_hoosh/lfw/{lin[i+1][0]}/{lin[i+1][0]}_{st}.txt', 'r')
  y = open(f'/content/gdrive/My Drive/payanterm_hoosh/lfw/{lin[i+1][2]}/{lin[i+1][2]}_{st1}.txt', 'r')
```

همین عملیات را برای داده های تست نیز تکرار میکنیم تا داده های تست نیز کامل شوند.

اما هم داده های تست هم داده های آموزشی دارای ابعاد مساوی نیستند. مثلا بعضی از داده ها دارای 4 سطر و بعضی دیگر دارای 5 سطر هستند. پس باید آنها را به داده های سطری تبدیل کرد. چون همه دادگان دارای 512 ویژگی (پیکسل) هستند، پس میتوان هر داده را به یک بردار 512 تایی تبدیل کرد که همه داده ها از نظر ابعاد مساوی شوند.

برای اینکه داده ها بصورت آرایه های دوبعدی تبدیل شوند، سعی میکنیم 512 پیکسل را بعنوان بعد دوم قرار دهیم که در تصویر زیر این تبدیلات مشخص شده اند.

```
flatten_list = lambda y:[x for a in y for x in flatten_list(a)] if type(y) is list else [y]
temp = flatten_list(Xtrain_first)
temp2 = flatten_list(Xtrain_second)
```

```
flatten_list = lambda y:[x for a in y for x in flatten_list(a)] if type(y) is list else [y]
temp3 = flatten_list(Xtest_first)
temp4 = flatten_list(Xtest_second)
```

```
import numpy as np
Xtrain_first = np.array(temp).reshape(-1, 512)
Xtrain_second = np.array(temp2).reshape(-1, 512)
```

```
Xtest_first = np.array(temp3).reshape(-1, 512)
Xtest_second = np.array(temp4).reshape(-1, 512)
```

حالا باید تصمیم بگیریم که داده های دسته اول که مربوط به شخص اول هستند، چگونه به داده های دسته دوم که مربوط به شخص دوم هستند، مرتبط شوند. راه ابتدایی که به ذهن من رسید این بود که داده های دسته اول را به داده های دسته دوم concatenate کنیم. در تصویر زیر نمایش آنرا میبینیم که از طریق axis = 1 این کار انجام شده است. بنابراین عکسهایی که قرار است باهم مقایسه شوند، در کنارهم قرار میگیرند.

```
import pandas as pd

df1 = pd.DataFrame(Xtrain_first)
df2 = pd.DataFrame(Xtrain_second)
print(df1.shape)
print(df2.shape)

df_contact = pd.concat([df1, df2], axis=1)
```

```
df3 = pd.DataFrame(Xtest_first)
df4 = pd.DataFrame(Xtest_second)
print(df3.shape)
print(df4.shape)

df_contact_test = pd.concat([df3, df4], axis=1)
```

قبل از اینکه به پیاده سازی الگوریتمها بپردازیم، میتوانیم از یک روش برای بهبود عملکردها استفاده کنیم. اگر داده های اموزشی و target آنها را باهم concatenate کنیم و سپس کل مجموعه را shuffle کنیم، باعث بهبود الگوریتم خواهد شد. چون داده ها بصورت پیش فرض به این صورت است که همه داده هایی که y = 1 هستند پشت سر هم است و همه داده هایی که y = 0 است پشت سرهم. این باعث عدم یادگیری صحیح خواهد شد. پس:

```
train = pd.concat([df_contact, df_y], ignore_index=True, sort=False, axis=1)
test = pd.concat([df_contact_test, df_y_test], ignore_index=True, sort=False, axis=1)
```

```
train = train.sample(frac = 1)
test = test.sample(frac = 1)
```

```
df_contact = train.loc[:, :1023]
df_y = train.loc[:, 1024]
df_contact_test = test.loc[:, :1023]
df_y_test = test.loc[:, 1024]
```

اکنون میخواهیم به بررسی الگوریتمهای مختلف روی این دیتاست بپردازیم. ابتدا از الگوریتم Bagging آغاز میکنیم. این الگوریتم سه پارامتر خیلی مهم دارد. میتوانیم آرگومان base_estimator را برابر DecisionTreeClassifier قرار دهیم. بنابراین پیش بینی کننده ی پایه ای درخت تصمیم خواهد بود. بسته به میزان مصرف زمان و حافظه، تعداد درختان را عوض میکنیم. در اینجا بهترین حالت که یک مصالحه ای برقرار کند را برابر 250 قرار میدهیم. عدد random_state هم برابر 7 یا اعداد دیگر میتوانیم قرار دهیم. پس تصویر زیر نمایشی از آن است:

```
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

model = BaggingClassifier(base_estimator = DecisionTreeClassifier(), n_estimators = 250, random_state=7)
model.fit(df_contact, df_y.ravel())
yhat = model.predict(df_contact_test)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(df_y_test, yhat)
```

```
0.689
```

مشخص است که دقت حدود 69 درصد برای این الگوریتم بدست آمده است.

سپس به الگوریتم AdaBoost میپردازیم. این مدل هم دو آرگومان مهم دارد. تعداد پیش بینی کننده ها را در این بخش برابر 1000 قرار میدهیم. نتیجه در تصویر مشخص است. اما دقت این الگوریتم، دقت بالایی نیست.

```
import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier

model = AdaBoostClassifier(n_estimators=1000, random_state=7)
model.fit(df_contact, df_y.ravel())
yhat = model.predict(df_contact_test)
```

```
from sklearn.metrics import accuracy_score
df_y_test = np.array(df_y_test)
accuracy_score(df_y_test, yhat)
```

```
0.522
```

سومین الگوریتم بسیار مهم در حوزه ensemble، الگوریتم random forest است. این الگوریتم زمان اجرای زیادی دارد. با تست کردن مقادیر مختلف برای تعداد پیش بینی کننده ها، از 100 تا 10000، به این نتیجه رسیدم که اگر تعداد پیش بینی کننده ها برابر 5000 باشد، بهترین نتیجه با صرف زمانی کمتر بدست می آید.

```
import pandas
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=10000, max_features=128)
model.fit(df_contact, df_y.ravel())
yhat = model.predict(df_contact_test)
```

```
from sklearn.metrics import accuracy_score
df_y_test_ = df_y_test
accuracy_score(df_y_test_, yhat)
```

```
0.726
```

اما اصلی ترین بخش برای الگوریتم های ensemble، الگوریتم هایی است که از آن بعنوان voting یاد میشود. این الگوریتم ها از نظر چند classifier استفاده میکنند. من در این بخش، از دو نوع الگوریتم voting استفاده خواهم کرد.

اولین مدل، از classifier های logisticRegression، DecisionTree و SVC استفاده میکند.

دومین مدل، از classifier های DecisionTree، KNeighbors و SVC استفاده میکند. در دومین مدل، برای هر Classifier یک وزن تعیین کرده ام. این وزنها با تکرار چندباره الگوریتم مشخص شده اند، طوریکه هر طبقه بندی که نتیجه بهتری بدهد، همان طبقه بند، وزن بهتری دارد.

```python
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

estimators = []
model1 = LogisticRegression()
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))

ensemble = VotingClassifier(estimators)
model.fit(df_contact, df_y.ravel())
yhat = model.predict(df_contact_test)
```

```python
from sklearn.metrics import accuracy_score
accuracy_score(df_y_test, yhat)
```

```
0.522
```

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from itertools import product
from sklearn.ensemble import VotingClassifier

clf1 = DecisionTreeClassifier(max_depth=4)
clf2 = KNeighborsClassifier(n_neighbors=7)
clf3 = SVC(kernel='rbf', probability=True)
eclf = VotingClassifier(estimators=[('dt', clf1), ('knn', clf2), ('svc', clf3)], voting='soft', weights=[1, 3, 5])

clf1.fit(df_contact, df_y.ravel())
yhat_1 = clf1.predict(df_contact_test)
clf2.fit(df_contact, df_y.ravel())
yhat_2 = clf2.predict(df_contact_test)
clf3.fit(df_contact, df_y.ravel())
yhat_3 = clf3.predict(df_contact_test)
eclf.fit(df_contact, df_y.ravel())
yhat_4 = eclf.predict(df_contact_test)
```

```
from sklearn.metrics import accuracy_score

print(accuracy_score(df_y_test, yhat_1))
print(accuracy_score(df_y_test, yhat_2))
print(accuracy_score(df_y_test, yhat_3))
print(accuracy_score(df_y_test, yhat_4))
```

```
0.523
0.682
0.732
0.735
```

پس تا اینجای کار، بیشترین دقت برای الگوریتم SVC با الگوریتم کرنل rbf ثبت شد که نتیجه ی آن هم در الگوریتم ensemble مشهود است.

الگوریتم boosting دیگری که میتوانیم برای classification استفاده کنیم، الگوریتم Hist Gradient Tree Boosting میباشد. این الگوریتم هم با توجه به اینکه زمان اجرای پایینی دارد و دقت بالایی ارائه میدهد، مدل مناسبی ایجاد میکند.

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import HistGradientBoostingClassifier

clf = HistGradientBoostingClassifier(loss='binary_crossentropy', learning_rate=0.2).fit(df_contact, df_y.ravel())
clf.score(df_contact_test, df_y_test)
```

```
0.734
```

در آخرین مرحله تصمیم گرفتم یک شبکه عصبی یک لایه انتخاب کنم و آنرا بعنوان یکی از classifier های الگوریتم voting استفاده کنم. بهترین دقت تا اینجای کار از شبکه عصبی بدست آمد. همانطور که مشخص است از adam بعنوان optimizer استفاده شده و نرخ یادگیری با سعی و خطا بدست آمده است.

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam', alpha=1e-2, hidden_layer_sizes=(128,), random_state=1)
clf.fit(df_contact, df_y.ravel())
yhat = clf.predict(df_contact_test)
accuracy_score(df_y_test, yhat)
```

```
0.774
```

استفاده در مدل voting:

```
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

clf1 = MLPClassifier(solver='adam', alpha=1e-2, hidden_layer_sizes=(128,), random_state=1)
clf2 = KNeighborsClassifier(n_neighbors=7)
clf3 = SVC(kernel='rbf', probability=True)
eclf = VotingClassifier(estimators=[('mlp', clf1), ('knn', clf2), ('svc', clf3)], voting='soft', weights=[5, 3, 4])

clf1.fit(df_contact, df_y.ravel())
yhat_1 = clf1.predict(df_contact_test)
clf2.fit(df_contact, df_y.ravel())
yhat_2 = clf2.predict(df_contact_test)
clf3.fit(df_contact, df_y.ravel())
yhat_3 = clf3.predict(df_contact_test)
eclf.fit(df_contact, df_y.ravel())
yhat_4 = eclf.predict(df_contact_test)
```

```
from sklearn.metrics import accuracy_score

print(accuracy_score(df_y_test, yhat_1))
print(accuracy_score(df_y_test, yhat_2))
print(accuracy_score(df_y_test, yhat_3))
print(accuracy_score(df_y_test, yhat_4))
```

```
0.774
0.682
0.732
0.786
```

در این مدل voting که از شبکه عصبی، KNeighbors و SVC استفاده کردیم، دقت به 78.6 درصد رسید.

اما مسئله در اینجا ختم نشد. من تلاش زیادی کردم که بتوانم درصد دقت را از این بالاتر ببرم. بنابراین نحوه تشکیل آرایه های تست و آموزش را مورد بازبینی قرار دادم. به این منظور، داده ی مجموعه اول که متعلق به نفر اول است را از داده ی مجموعه دوم که متعلق به نفر دوم است کم کردم. یعنی داده ی 512 تایی از مجموعه اول را از داده ی 512 تایی مجموعه دوم کم کردم و یک بردار 512 تایی بدست آمد که حاصل تفریق آن دو بود. این شکل از دادگان دو مزیت دارد.

اولا مفهوم distance در آن دیده میشود. پس مدل با داشتن فاصله دو تصویر، سعی میکند الگوریتمی برای یافتن پاسخ بیابد. ثانیا تعداد ویژگی ها نسبت به حالت قبل نصف شده است. این به بهبود نتیجه کمک فراوانی خواهد کرد.

```
fd_concat_distance = np.zeros((df_contact.shape[0], int(df_contact.shape[1]/2)))
c = 0
for row in df_contact:
  for element in range(row.shape[0]):
    if element < 512:
      fd_concat_distance[c][element] = row[element] - row[element + 512]
  c = c + 1

fd_concat_distance.shape
```

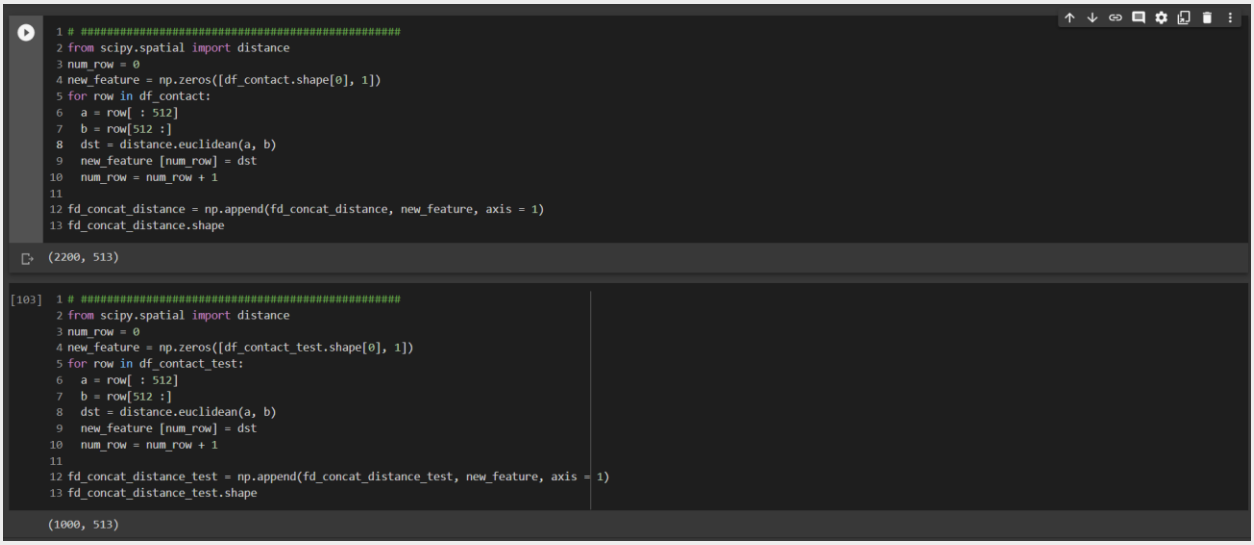
```
(2200, 512)
```

```
fd_concat_distance_test = np.zeros((df_contact_test.shape[0], int(df_contact_test.shape[1]/2)))
c = 0
for row in df_contact_test:
  for element in range(row.shape[0]):
    if element < 512:
      fd_concat_distance_test[c][element] = row[element] - row[element + 512]
  c = c + 1

fd_concat_distance_test.shape
```

```
(1000, 512)
```

در این راستا، میتوانیم از روشهای دیگری نیز استفاده کنیم. مثلا میتوانیم سه ویژگی به داده هایمان اضافه کنیم که فاصله اقلیدسی، فاصله منهتن و فاصله کسینوسی است. در تصاویر زیر، پیاده سازی ان انجام شده است.



```
# ##############################################
from scipy.spatial import distance
num_row = 0
new_feature = np.zeros([df_contact.shape[0], 1])
for row in df_contact:
  a = row[ : 512]
  b = row[512 :]
  dst = distance.euclidean(a, b)
  new_feature [num_row] = dst
  num_row = num_row + 1

fd_concat_distance = np.append(fd_concat_distance, new_feature, axis = 1)
fd_concat_distance.shape
```

```
(2200, 513)
```

```
[103]
# ##############################################
from scipy.spatial import distance
num_row = 0
new_feature = np.zeros([df_contact_test.shape[0], 1])
for row in df_contact_test:
  a = row[ : 512]
  b = row[512 :]
  dst = distance.euclidean(a, b)
  new_feature [num_row] = dst
  num_row = num_row + 1

fd_concat_distance_test = np.append(fd_concat_distance_test, new_feature, axis = 1)
fd_concat_distance_test.shape
```

```
(1000, 513)
```

```
##############################################
from sklearn.metrics.pairwise import manhattan_distances
num_row = 0
new_feature = np.zeros([df_contact.shape[0], 1])
for row in df_contact:
  a = row[ : 512].reshape(1, -1)
  b = row[512 :].reshape(1, -1)
  dst = manhattan_distances(a, b)
  new_feature [num_row] = dst
  num_row = num_row + 1

fd_concat_distance = np.append(fd_concat_distance, new_feature, axis = 1)
fd_concat_distance.shape
```

```
(2200, 513)
```

```
[125] ##############################################
from sklearn.metrics.pairwise import manhattan_distances
num_row = 0
new_feature = np.zeros([df_contact_test.shape[0], 1])
for row in df_contact_test:
  a = row[ : 512].reshape(1, -1)
  b = row[512 :].reshape(1, -1)
  dst = manhattan_distances(a, b)
  new_feature [num_row] = dst
  num_row = num_row + 1

fd_concat_distance_test = np.append(fd_concat_distance_test, new_feature, axis = 1)
fd_concat_distance_test.shape
```

```
(1000, 513)
```

```
##############################################
from sklearn.metrics.pairwise import cosine_similarity
num_row = 0
new_feature = np.zeros([df_contact.shape[0], 1])
for row in df_contact:
  a = row[ : 512].reshape(1, -1)
  b = row[512 :].reshape(1, -1)
  dst = manhattan_distances(a, b)
  new_feature [num_row] = dst
  num_row = num_row + 1

fd_concat_distance = np.append(fd_concat_distance, new_feature, axis = 1)
fd_concat_distance.shape
```

```
(2200, 514)
```

```
[127] ##############################################
from sklearn.metrics.pairwise import cosine_similarity
num_row = 0
new_feature = np.zeros([df_contact_test.shape[0], 1])
for row in df_contact_test:
  a = row[ : 512].reshape(1, -1)
  b = row[512 :].reshape(1, -1)
  dst = manhattan_distances(a, b)
  new_feature [num_row] = dst
  num_row = num_row + 1

fd_concat_distance_test = np.append(fd_concat_distance_test, new_feature, axis = 1)
fd_concat_distance_test.shape
```

```
(1000, 514)
```

البته انجام این تغییرات تاثیری در خروجی نداشت. من جستجو کردم و فهمیدم که تابع Euclidean که در کتابخانه distance تعریف شده، دلیل این کم بودن دقت است. بنابراین از کتابخانه sklearn استفاده کردم و نتیجه به بهترین حالت رسید.

```
# ################################################
from sklearn.metrics.pairwise import euclidean_distances
num_row = 0
new_feature = np.zeros([df_contact.shape[0], 1])
for row in df_contact:
  a = row[ : 512].reshape(1, -1)
  b = row[512 :].reshape(1, -1)
  dst = euclidean_distances(a, b)
  new_feature [num_row] = dst
  num_row = num_row + 1

fd_concat_distance = np.append(fd_concat_distance, new_feature, axis = 1)
fd_concat_distance.shape
```

```
(2200, 513)
```

```
# ################################################
from sklearn.metrics.pairwise import euclidean_distances
num_row = 0
new_feature = np.zeros([df_contact_test.shape[0], 1])
for row in df_contact_test:
  a = row[ : 512].reshape(1, -1)
  b = row[512 :].reshape(1, -1)
  dst = euclidean_distances(a, b)
  new_feature [num_row] = dst
  num_row = num_row + 1

fd_concat_distance_test = np.append(fd_concat_distance_test, new_feature, axis = 1)
fd_concat_distance_test.shape
```

```
(1000, 513)
```

یک مرحله دیگر که حتما باید برروی دادگان انجام شود، بحث استاندارد سازی است. این فرایند را توسط تابع standardScalar انجام میدهیم.

Standard

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
fd_concat_distance = sc.fit_transform(fd_concat_distance)
fd_concat_distance_test = sc.transform (fd_concat_distance_test)
```

حالا به نتایج تست مدلهای مختلف روی این دادگان میپردازیم.

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam', alpha=1e-1, hidden_layer_sizes=(300,), random_state=1)
clf.fit(fd_concat_distance, df_y.ravel())
yhat = clf.predict(fd_concat_distance_test)
accuracy_score(df_y_test, yhat)
```

```
0.88
```

میبینیم که مدل شبکه عصبی با نرخ یادگیری 0.1 و تعداد 300 نورون، 88 درصد خروجی میدهد که بهترین دقت در بین مدلهایی است که تاکنون ران شده است.

حالا از مدل voting استفاده میکنیم که مدل ensemble است.

```
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

clf1 = MLPClassifier(solver='adam', alpha=1e-1, hidden_layer_sizes=(300,), random_state=1)
clf2 = SVC(kernel='rbf', probability=True)
eclf = VotingClassifier(estimators=[('mlp', clf1), ('svc', clf2)], voting='soft', weights=[5, 4])

clf1.fit(fd_concat_distance, df_y.ravel())
yhat_1 = clf1.predict(fd_concat_distance_test)
clf2.fit(fd_concat_distance, df_y.ravel())
yhat_2 = clf2.predict(fd_concat_distance_test)
eclf.fit(fd_concat_distance, df_y.ravel())
yhat_4 = eclf.predict(fd_concat_distance_test)
```

```
from sklearn.metrics import accuracy_score

print(accuracy_score(df_y_test, yhat_1))
print(accuracy_score(df_y_test, yhat_2))
print(accuracy_score(df_y_test, yhat_4))
```

```
0.88
0.859
0.88
```

برای بهبود کارایی الگوریتم یک ایده ای که میتوان مطرح کرد، اضافه کردن داده های آموزشی است. برای این منظور، فرض میکنیم تصاویر آموزشی تاریک تر باشند، یا روشن تر باشند. بنابراین میتوان به ازای هر تصویر، دو تصویر جدید بوجود آورد که تاریکتر یا روشنتر از تصویر اولیه باشند. برای این منظور، بخش darker و lighter کد زده شده اند. من بعنوان نمونه، همه پیکسلهای این تصاویر را 0.1 تیره تر یا روشنتر کرده ام.

Darker

```
print(df_norm_.shape)
num_row = 0
for row in fd_concat_distance:
  darker = np.empty([0,0])
  for elmnt in range(row.shape[0]-3):
    temp = row[elmnt] + 0.05
    if temp > 1:
      temp = 1
    darker = np.append(darker, temp)

  darker  = np.append(darker, row[512])
  darker  = np.append(darker, row[513])
  darker  = np.append(darker, row[514])
  darker = np.expand_dims(darker, axis=1)
  df_norm_ = np.concatenate((df_norm_, darker.T), axis=0)
  y_ = np.append(y_, y_[num_row])
  num_row = num_row + 1
print(df_norm_.shape)
```

```
(2200, 515)
(4400, 515)
```

Lighter

```
print(df_norm_.shape)
num_row = 0
for row in fd_concat_distance:
  lighter = np.empty([0,0])
  for elmnt in range(row.shape[0] - 3):
    temp = row[elmnt] - 0.05
    if temp < 0:
      temp = 0
    lighter = np.append(lighter, temp)

  lighter  = np.append(lighter, row[512])
  lighter  = np.append(lighter, row[513])
  lighter  = np.append(lighter, row[514])
  lighter = np.expand_dims(lighter, axis=1)
  df_norm_ = np.concatenate((df_norm_, lighter.T), axis=0)
  y_ = np.append(y_, y_[num_row])
  num_row = num_row + 1
print(df_norm_.shape)
```

```
(4400, 515)
(6600, 515)
```

پس از آن، الگوی rotate با 180 درجه زاویه را پیاده سازی کردم ولی نتیجه چندان جالب نشد. تصویر آن بصورت زیر است:

Rotate

```
print(df_norm_.shape)
num_row = 0
for row in fd_concat_distance:

  swaper = np.empty([0,0])
  for elmnt in range(row.shape[0]):
    temp = row[row.shape[0] - elmnt - 1]
    swaper = np.append(swaper, temp)

  swaper = np.expand_dims(swaper, axis=1)
  df_norm_ = np.concatenate((df_norm_, swaper.T), axis=0)
  df_y = np.append(df_y, df_y[num_row])
  num_row = num_row + 1
print(df_norm_.shape)
```

اما فیلتر salt and pepper فیلتر مناسبی برای تصاویر است. در این فیلتر، بصورت رندم بعضی از پیکسلها را سیاه و بعضی را سفید میکنیم. پس یعنی بعضی از پیکسلها 0 و بعضی دیگر 1 خواهند شد. در این کد 32 پیکسل بصورت رندم برای هر بردار 512 تایی انتخاب میشود.

```python
print(df_norm_.shape)
num_row = 0
for row in fd_concat_distance:

  salt_pepper = np.empty([0,0])
  randm_dark = np.random.randint(512, size=32)
  randm_light = np.random.randint(512, size=32)

  for elmnt in range(row.shape[0]):
    if elmnt in randm_dark and elmnt not in [512, 513, 514]:
      temp = df_norm_.min()
      salt_pepper = np.append(salt_pepper, temp)

    elif elmnt in randm_light and elmnt not in [512, 513, 514]:
      temp = df_norm_.max()
      salt_pepper = np.append(salt_pepper, temp)

    else:
      temp = row[elmnt]
      salt_pepper = np.append(salt_pepper, temp)

  salt_pepper = np.expand_dims(salt_pepper, axis=1)
  df_norm_ = np.concatenate((df_norm_, salt_pepper.T), axis=0)
  y_ = np.append(y_, y_[num_row])
  num_row = num_row + 1

print(df_norm_.shape)
```

```
(2200, 515)
(4400, 515)
```

اگر هرکدام از فیلترها را بصورت جداگانه پیاده کنیم، نتایج متفاوتی خواهیم دید. اما برای مثال، پس از پیاده سازی salt and pepper نتیجه ی زیر بدست می اید.

Test

```
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

clf1 = MLPClassifier(solver='adam', alpha=1e-1, hidden_layer_sizes=(300,), random_state=1)
clf2 = SVC(kernel='rbf', probability=True)
eclf = VotingClassifier(estimators=[('mlp', clf1), ('svc', clf2)], voting='soft', weights=[5, 4])

clf1.fit(df_norm_, y_.ravel())
yhat_1 = clf1.predict(fd_concat_distance_test)
clf2.fit(df_norm_, y_.ravel())
yhat_2 = clf2.predict(fd_concat_distance_test)
eclf.fit(df_norm_, y_.ravel())
yhat_4 = eclf.predict(fd_concat_distance_test)
```

```
[284] from sklearn.metrics import accuracy_score

print(accuracy_score(df_y_test, yhat_1))
print(accuracy_score(df_y_test, yhat_2))
print(accuracy_score(df_y_test, yhat_4))
```

```
0.867
0.839
0.871
```

برای مثال دیگر از درخت تصمیم استفاده میکنیم. مشاهده میکنیم که دقت آن با توجه به پردازش های صورت گرفته بشدت بالا رفته است.

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0, criterion='entropy', class_weight='balanced')
clf.fit(fd_concat_distance, df_y.ravel())
yhat_dt = clf.predict(fd_concat_distance_test)
print(accuracy_score(df_y_test, yhat_dt))
```

```
0.808
```

```python
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.svm import SVC

clf1 = DecisionTreeClassifier(random_state=0, criterion='entropy', class_weight='balanced')
clf2 = RandomForestClassifier(n_estimators=500, random_state=1)
clf3 = GaussianNB()
clf1.fit(fd_concat_distance, df_y.ravel())
clf2.fit(fd_concat_distance, df_y.ravel())
clf3.fit(fd_concat_distance, df_y.ravel())

yhat_1 = clf1.predict(fd_concat_distance_test)
yhat_2 = clf2.predict(fd_concat_distance_test)
yhat_3 = clf3.predict(fd_concat_distance_test)
```

```python
from sklearn.metrics import accuracy_score
print(accuracy_score(df_y_test, yhat_1))
print(accuracy_score(df_y_test, yhat_2))
print(accuracy_score(df_y_test, yhat_3))

eclf1 = VotingClassifier(estimators=[('DecisionTreeClassifier', clf1), ('RandomForestClassifier', clf2), ('GaussianNB', clf3)],
                         voting='soft', weights=[5, 5, 4])
eclf1 = eclf1.fit(fd_concat_distance, df_y.ravel())
yhat_res = eclf1.predict(fd_concat_distance_test)
print(accuracy_score(df_y_test, yhat_1))
```

```
0.808
0.861
0.8
0.808
```

در مرحله بعدی میتوانیم از لایه pooling در شبکه CNN الهام بگیریم. برای این منظور، هر چهار پیکسل کنار هم را درنظر بگیرید که چهار خانه مجاور در آرایه میباشد. فرض کنیم اندیس آنها بین i تا i+3 است. از این چهار عدد میانگین میگیریم و آن را در خانه ی i ام بردار جدیدی قرار میدهیم. سپس i را یکی زیاد میکنیم تا به انتهای ارایه برسیم. این بردارها را به انتهای آرایه داده های اموزشی اضافه میکنیم. کد آن بصورت زیر است و نتیجه متعاقبا آورده شده است.

```
print(df_norm.shape)
num_row = 0
for row in fd_concat_distance:
  mean_row = np.empty([0,0])
  elmnt = 0
  while elmnt < df_norm.shape[1]:
    if elmnt + 3 < 512:
      li = [row[elmnt], row[elmnt+1], row[elmnt+2], row[elmnt+3]]
      temp = sum(li) / 4
      mean_row = np.append(mean_row, temp)
      elmnt = elmnt + 1
    else:
      temp = row[elmnt]
      mean_row = np.append(mean_row, temp)
      elmnt = elmnt + 1

  mean_row = np.expand_dims(mean_row, axis=1)
  df_norm = np.concatenate((df_norm, mean_row.T), axis=0)
  y_ = np.append(y_, y_[num_row])
  num_row = num_row + 1
print(df_norm.shape)
```

```
(2200, 515)
(4400, 515)
```

```
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

clf1 = MLPClassifier(solver='adam', alpha=1e-1, hidden_layer_sizes=(300,), random_state=1)
clf2 = SVC(kernel='rbf', probability=True)
eclf = VotingClassifier(estimators=[('mlp', clf1), ('svc', clf2)], voting='soft', weights=[5, 4])

clf1.fit(df_norm, y_.ravel())
yhat_1 = clf1.predict(df_norm_test)
clf2.fit(df_norm, y_.ravel())
yhat_2 = clf2.predict(df_norm_test)
eclf.fit(df_norm, y_.ravel())
yhat_4 = eclf.predict(df_norm_test)
```

```
from sklearn.metrics import accuracy_score

print(accuracy_score(df_y_test, yhat_1))
print(accuracy_score(df_y_test, yhat_2))
print(accuracy_score(df_y_test, yhat_4))
```

```
0.865
0.869
0.872
```

در این گام، تصمیم گرفتم که به نحوی تعداد فیچرها را کاهش دهم. برای اینکار از PCA استفاده کردم. کد آن بصورت زیر است.

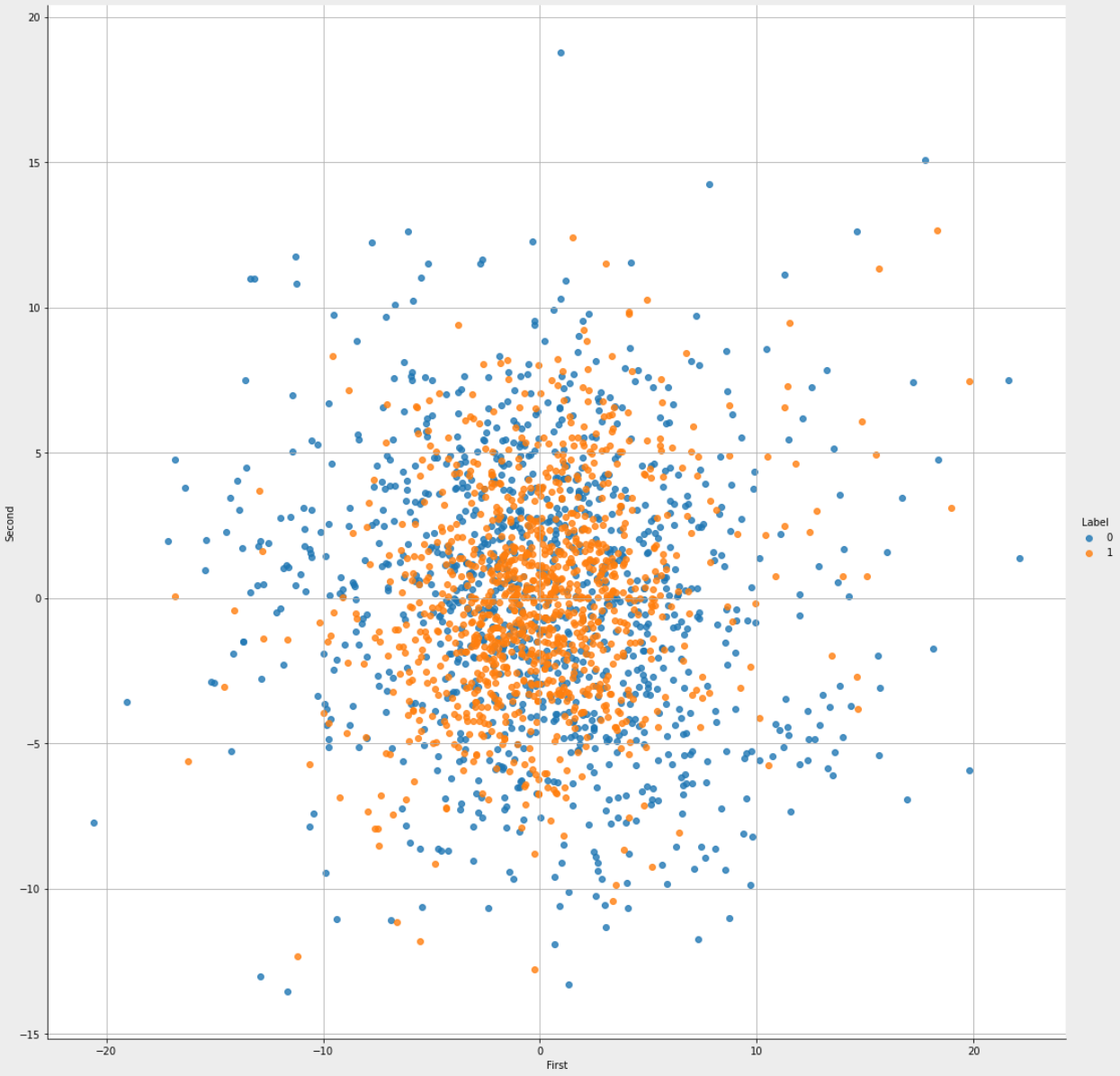
```
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
pca = PCA(n_components= 256, random_state=7, svd_solver = 'arpack')
X_PCA = pca.fit_transform(df_norm)
X_PCA = pd.DataFrame(data = X_PCA)

X_PCA_test = pca.fit_transform(df_norm_test)
X_PCA_test = pd.DataFrame(data = X_PCA_test)
```

```
X_PCA.shape
```

```
(2200, 256)
```

تصویر زیر، نموداری از دو فیچر اول داده های انتقال داده شده است.



مشاهده میشود که تعداد فیچرها را برابر 256 یعنی نصف حالت قبل قرار دادیم. با این توضیح، آن را تست میکنیم. اما متاسفانه الگوریتم بسیار بد عمل میکند.

```python
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

clf1 = MLPClassifier(solver='adam', alpha=1e-4, hidden_layer_sizes=(128,), random_state=1)
clf2 = SVC(kernel='rbf', probability=True)
eclf = VotingClassifier(estimators=[('mlp', clf1), ('svc', clf2)], voting='soft', weights=[5, 4])

clf1.fit(X_PCA, y_.ravel())
yhat_1 = clf1.predict(X_PCA_test)
clf2.fit(X_PCA, y_.ravel())
yhat_2 = clf2.predict(X_PCA_test)
eclf.fit(X_PCA, y_.ravel())
yhat_4 = eclf.predict(X_PCA_test)
```

```python
from sklearn.metrics import accuracy_score

print(accuracy_score(df_y_test, yhat_1))
print(accuracy_score(df_y_test, yhat_2))
print(accuracy_score(df_y_test, yhat_4))
```

```
0.538
0.703
0.566
```

یک راه دیگر که برای کم کردن تعداد فیچرها میشد استفاده کرد این بود که دو پیکسل از تصویر اول را با دو پیکسل از تصویر دوم که قرار بود با هم مقایسه شوند در نظر بگیریم. اکنون دو بردار دوعضوی داریم. سپس با فاصله اقلیدسی آنها فاصله آنها را حساب کنیم. این کار را ادامه میدهیم و سپس یک آرایه ای با نصف فیچرهای قبلی خواهیم داشت. کد آن بصورت زیر است. این کار روی داده های تست نیز انجام میشود.

```python
from sklearn.metrics.pairwise import euclidean_distances
df_norm_ = np.zeros((df_contact.shape[0], int(df_contact.shape[1]/4)))
c = 0
for row in df_contact:
  p = 0
  it = 0
  while p < (row.shape[0]-3)/2:
    a = np.array((row[p], row[p+1])).reshape(1, -1)
    b = np.array((row[p+512], row[p+513])).reshape(1, -1)
    dst = euclidean_distances(a, b)
    df_norm_[c][it] = dst
    it = it + 1
    p = p + 2

  c = c + 1

df_norm_.shape
```

```
(2200, 256)
```

اما این روش کاهش ویژگی هم به خوبی موفق نبود.

```python
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

clf1 = MLPClassifier(solver='adam', alpha=1e-2, hidden_layer_sizes=(128,), random_state=1)
clf2 = SVC(kernel='rbf', probability=True)
eclf = VotingClassifier(estimators=[('mlp', clf1), ('svc', clf2)], voting='soft', weights=[5, 5])

clf1.fit(df_norm_, y_.ravel())
yhat_1 = clf1.predict(df_norm_test_)
clf2.fit(df_norm_, y_.ravel())
yhat_2 = clf2.predict(df_norm_test_)
eclf.fit(df_norm_, y_.ravel())
yhat_4 = eclf.predict(df_norm_test_)
```

```python
from sklearn.metrics import accuracy_score

print(accuracy_score(df_y_test, yhat_1))
print(accuracy_score(df_y_test, yhat_2))
print(accuracy_score(df_y_test, yhat_4))
```

```
0.704
0.838
0.768
```

بعنوان جمع بندی، بهترین دقت ***88*** درصد ثبت شد.

کامیار نصیری