Here are the packages we use.

```
import time

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import fetch_openml
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier
from sklearn.decomposition import PCA
from sklearn.decomposition import IncrementalPCA
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import cross_val_predict



#mnist = fetch_openml('mnist_784')
mnist = fetch_openml('mnist_784', version = 1, as_frame = False) #this as_frame = False wil
#helps makes sense of X[0] in some_digit defintion.
```

```
mnist
```

```
{'data': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]),
 'target': array(['5', '0', '4', ..., '4', '5', '6'], dtype=object),
 'frame': None,
 'categories': {},
 'feature_names': ['pixel1',
  'pixel2',
  'pixel3',
  'pixel4',
  'pixel5',
  'pixel6',
  'pixel7',
  'pixel8',
  'pixel9',
```

```
'pixel10',
'pixel11',
'pixel12',
'pixel13',
'pixel14',
'pixel15',
'pixel16',
'pixel17',
'pixel18',
'pixel19',
'pixel20',
'pixel21',
'pixel22',
'pixel23',
'pixel24',
'pixel25',
'pixel26',
'pixel27',
'pixel28',
'pixel29',
'pixel30',
'pixel31',
'pixel32',
'pixel33',
'pixel34',
'pixel35',
'pixel36',
'pixel37',
'pixel38',
'pixel39',
'pixel40',
'pixel41',
'pixel42',
'pixel43',
'pixel44',
'pixel45',
'pixel46',
'pixel47',
'pixel48',
'pixel49',
'pixel50',
'pixel51',
'pixel52',
'pixel53',
'pixel54',
'pixel55',
```

```
'pixel56',
'pixel57',
'pixel58',
'pixel59',
'pixel60',
'pixel61',
'pixel62',
'pixel63',
'pixel64',
'pixel65',
'pixel66',
'pixel67',
'pixel68',
'pixel69',
'pixel70',
'pixel71',
'pixel72',
'pixel73',
'pixel74',
'pixel75',
'pixel76',
'pixel77',
'pixel78',
'pixel79',
'pixel80',
'pixel81',
'pixel82',
'pixel83',
'pixel84',
'pixel85',
'pixel86',
'pixel87',
'pixel88',
'pixel89',
'pixel90',
'pixel91',
'pixel92',
'pixel93',
'pixel94',
'pixel95',
'pixel96',
'pixel97',
'pixel98',
'pixel99',
'pixel100',
'pixel101',
```

```
'pixel102',
'pixel103',
'pixel104',
'pixel105',
'pixel106',
'pixel107',
'pixel108',
'pixel109',
'pixel110',
'pixel111',
'pixel112',
'pixel113',
'pixel114',
'pixel115',
'pixel116',
'pixel117',
'pixel118',
'pixel119',
'pixel120',
'pixel121',
'pixel122',
'pixel123',
'pixel124',
'pixel125',
'pixel126',
'pixel127',
'pixel128',
'pixel129',
'pixel130',
'pixel131',
'pixel132',
'pixel133',
'pixel134',
'pixel135',
'pixel136',
'pixel137',
'pixel138',
'pixel139',
'pixel140',
'pixel141',
'pixel142',
'pixel143',
'pixel144',
'pixel145',
'pixel146',
'pixel147',
```

```
'pixel148',
'pixel149',
'pixel150',
'pixel151',
'pixel152',
'pixel153',
'pixel154',
'pixel155',
'pixel156',
'pixel157',
'pixel158',
'pixel159',
'pixel160',
'pixel161',
'pixel162',
'pixel163',
'pixel164',
'pixel165',
'pixel166',
'pixel167',
'pixel168',
'pixel169',
'pixel170',
'pixel171',
'pixel172',
'pixel173',
'pixel174',
'pixel175',
'pixel176',
'pixel177',
'pixel178',
'pixel179',
'pixel180',
'pixel181',
'pixel182',
'pixel183',
'pixel184',
'pixel185',
'pixel186',
'pixel187',
'pixel188',
'pixel189',
'pixel190',
'pixel191',
'pixel192',
'pixel193',
```

```
'pixel194',
'pixel195',
'pixel196',
'pixel197',
'pixel198',
'pixel199',
'pixel200',
'pixel201',
'pixel202',
'pixel203',
'pixel204',
'pixel205',
'pixel206',
'pixel207',
'pixel208',
'pixel209',
'pixel210',
'pixel211',
'pixel212',
'pixel213',
'pixel214',
'pixel215',
'pixel216',
'pixel217',
'pixel218',
'pixel219',
'pixel220',
'pixel221',
'pixel222',
'pixel223',
'pixel224',
'pixel225',
'pixel226',
'pixel227',
'pixel228',
'pixel229',
'pixel230',
'pixel231',
'pixel232',
'pixel233',
'pixel234',
'pixel235',
'pixel236',
'pixel237',
'pixel238',
'pixel239',
```

```
'pixel240',
'pixel241',
'pixel242',
'pixel243',
'pixel244',
'pixel245',
'pixel246',
'pixel247',
'pixel248',
'pixel249',
'pixel250',
'pixel251',
'pixel252',
'pixel253',
'pixel254',
'pixel255',
'pixel256',
'pixel257',
'pixel258',
'pixel259',
'pixel260',
'pixel261',
'pixel262',
'pixel263',
'pixel264',
'pixel265',
'pixel266',
'pixel267',
'pixel268',
'pixel269',
'pixel270',
'pixel271',
'pixel272',
'pixel273',
'pixel274',
'pixel275',
'pixel276',
'pixel277',
'pixel278',
'pixel279',
'pixel280',
'pixel281',
'pixel282',
'pixel283',
'pixel284',
'pixel285',
```

```
'pixel286',
'pixel287',
'pixel288',
'pixel289',
'pixel290',
'pixel291',
'pixel292',
'pixel293',
'pixel294',
'pixel295',
'pixel296',
'pixel297',
'pixel298',
'pixel299',
'pixel300',
'pixel301',
'pixel302',
'pixel303',
'pixel304',
'pixel305',
'pixel306',
'pixel307',
'pixel308',
'pixel309',
'pixel310',
'pixel311',
'pixel312',
'pixel313',
'pixel314',
'pixel315',
'pixel316',
'pixel317',
'pixel318',
'pixel319',
'pixel320',
'pixel321',
'pixel322',
'pixel323',
'pixel324',
'pixel325',
'pixel326',
'pixel327',
'pixel328',
'pixel329',
'pixel330',
'pixel331',
```

```
'pixel332',
'pixel333',
'pixel334',
'pixel335',
'pixel336',
'pixel337',
'pixel338',
'pixel339',
'pixel340',
'pixel341',
'pixel342',
'pixel343',
'pixel344',
'pixel345',
'pixel346',
'pixel347',
'pixel348',
'pixel349',
'pixel350',
'pixel351',
'pixel352',
'pixel353',
'pixel354',
'pixel355',
'pixel356',
'pixel357',
'pixel358',
'pixel359',
'pixel360',
'pixel361',
'pixel362',
'pixel363',
'pixel364',
'pixel365',
'pixel366',
'pixel367',
'pixel368',
'pixel369',
'pixel370',
'pixel371',
'pixel372',
'pixel373',
'pixel374',
'pixel375',
'pixel376',
'pixel377',
```

```
'pixel378',
'pixel379',
'pixel380',
'pixel381',
'pixel382',
'pixel383',
'pixel384',
'pixel385',
'pixel386',
'pixel387',
'pixel388',
'pixel389',
'pixel390',
'pixel391',
'pixel392',
'pixel393',
'pixel394',
'pixel395',
'pixel396',
'pixel397',
'pixel398',
'pixel399',
'pixel400',
'pixel401',
'pixel402',
'pixel403',
'pixel404',
'pixel405',
'pixel406',
'pixel407',
'pixel408',
'pixel409',
'pixel410',
'pixel411',
'pixel412',
'pixel413',
'pixel414',
'pixel415',
'pixel416',
'pixel417',
'pixel418',
'pixel419',
'pixel420',
'pixel421',
'pixel422',
'pixel423',
```

```
'pixel424',
'pixel425',
'pixel426',
'pixel427',
'pixel428',
'pixel429',
'pixel430',
'pixel431',
'pixel432',
'pixel433',
'pixel434',
'pixel435',
'pixel436',
'pixel437',
'pixel438',
'pixel439',
'pixel440',
'pixel441',
'pixel442',
'pixel443',
'pixel444',
'pixel445',
'pixel446',
'pixel447',
'pixel448',
'pixel449',
'pixel450',
'pixel451',
'pixel452',
'pixel453',
'pixel454',
'pixel455',
'pixel456',
'pixel457',
'pixel458',
'pixel459',
'pixel460',
'pixel461',
'pixel462',
'pixel463',
'pixel464',
'pixel465',
'pixel466',
'pixel467',
'pixel468',
'pixel469',
```

```
'pixel470',
'pixel471',
'pixel472',
'pixel473',
'pixel474',
'pixel475',
'pixel476',
'pixel477',
'pixel478',
'pixel479',
'pixel480',
'pixel481',
'pixel482',
'pixel483',
'pixel484',
'pixel485',
'pixel486',
'pixel487',
'pixel488',
'pixel489',
'pixel490',
'pixel491',
'pixel492',
'pixel493',
'pixel494',
'pixel495',
'pixel496',
'pixel497',
'pixel498',
'pixel499',
'pixel500',
'pixel501',
'pixel502',
'pixel503',
'pixel504',
'pixel505',
'pixel506',
'pixel507',
'pixel508',
'pixel509',
'pixel510',
'pixel511',
'pixel512',
'pixel513',
'pixel514',
'pixel515',
```

```
'pixel516',
'pixel517',
'pixel518',
'pixel519',
'pixel520',
'pixel521',
'pixel522',
'pixel523',
'pixel524',
'pixel525',
'pixel526',
'pixel527',
'pixel528',
'pixel529',
'pixel530',
'pixel531',
'pixel532',
'pixel533',
'pixel534',
'pixel535',
'pixel536',
'pixel537',
'pixel538',
'pixel539',
'pixel540',
'pixel541',
'pixel542',
'pixel543',
'pixel544',
'pixel545',
'pixel546',
'pixel547',
'pixel548',
'pixel549',
'pixel550',
'pixel551',
'pixel552',
'pixel553',
'pixel554',
'pixel555',
'pixel556',
'pixel557',
'pixel558',
'pixel559',
'pixel560',
'pixel561',
```

```
'pixel562',
'pixel563',
'pixel564',
'pixel565',
'pixel566',
'pixel567',
'pixel568',
'pixel569',
'pixel570',
'pixel571',
'pixel572',
'pixel573',
'pixel574',
'pixel575',
'pixel576',
'pixel577',
'pixel578',
'pixel579',
'pixel580',
'pixel581',
'pixel582',
'pixel583',
'pixel584',
'pixel585',
'pixel586',
'pixel587',
'pixel588',
'pixel589',
'pixel590',
'pixel591',
'pixel592',
'pixel593',
'pixel594',
'pixel595',
'pixel596',
'pixel597',
'pixel598',
'pixel599',
'pixel600',
'pixel601',
'pixel602',
'pixel603',
'pixel604',
'pixel605',
'pixel606',
'pixel607',
```

```
'pixel608',
'pixel609',
'pixel610',
'pixel611',
'pixel612',
'pixel613',
'pixel614',
'pixel615',
'pixel616',
'pixel617',
'pixel618',
'pixel619',
'pixel620',
'pixel621',
'pixel622',
'pixel623',
'pixel624',
'pixel625',
'pixel626',
'pixel627',
'pixel628',
'pixel629',
'pixel630',
'pixel631',
'pixel632',
'pixel633',
'pixel634',
'pixel635',
'pixel636',
'pixel637',
'pixel638',
'pixel639',
'pixel640',
'pixel641',
'pixel642',
'pixel643',
'pixel644',
'pixel645',
'pixel646',
'pixel647',
'pixel648',
'pixel649',
'pixel650',
'pixel651',
'pixel652',
'pixel653',
```

```
'pixel654',
'pixel655',
'pixel656',
'pixel657',
'pixel658',
'pixel659',
'pixel660',
'pixel661',
'pixel662',
'pixel663',
'pixel664',
'pixel665',
'pixel666',
'pixel667',
'pixel668',
'pixel669',
'pixel670',
'pixel671',
'pixel672',
'pixel673',
'pixel674',
'pixel675',
'pixel676',
'pixel677',
'pixel678',
'pixel679',
'pixel680',
'pixel681',
'pixel682',
'pixel683',
'pixel684',
'pixel685',
'pixel686',
'pixel687',
'pixel688',
'pixel689',
'pixel690',
'pixel691',
'pixel692',
'pixel693',
'pixel694',
'pixel695',
'pixel696',
'pixel697',
'pixel698',
'pixel699',
```

```
'pixel700',
'pixel701',
'pixel702',
'pixel703',
'pixel704',
'pixel705',
'pixel706',
'pixel707',
'pixel708',
'pixel709',
'pixel710',
'pixel711',
'pixel712',
'pixel713',
'pixel714',
'pixel715',
'pixel716',
'pixel717',
'pixel718',
'pixel719',
'pixel720',
'pixel721',
'pixel722',
'pixel723',
'pixel724',
'pixel725',
'pixel726',
'pixel727',
'pixel728',
'pixel729',
'pixel730',
'pixel731',
'pixel732',
'pixel733',
'pixel734',
'pixel735',
'pixel736',
'pixel737',
'pixel738',
'pixel739',
'pixel740',
'pixel741',
'pixel742',
'pixel743',
'pixel744',
'pixel745',
```

```
 'pixel746',
 'pixel747',
 'pixel748',
 'pixel749',
 'pixel750',
 'pixel751',
 'pixel752',
 'pixel753',
 'pixel754',
 'pixel755',
 'pixel756',
 'pixel757',
 'pixel758',
 'pixel759',
 'pixel760',
 'pixel761',
 'pixel762',
 'pixel763',
 'pixel764',
 'pixel765',
 'pixel766',
 'pixel767',
 'pixel768',
 'pixel769',
 'pixel770',
 'pixel771',
 'pixel772',
 'pixel773',
 'pixel774',
 'pixel775',
 'pixel776',
 'pixel777',
 'pixel778',
 'pixel779',
 'pixel780',
 'pixel781',
 'pixel782',
 'pixel783',
 'pixel784'],
'target_names': ['class'],
'DESCR': "**Author**: Yann LeCun, Corinna Cortes, Christopher J.C. Burges  \n**Source**: [N
'details': {'id': '554',
 'name': 'mnist_784',
 'version': '1',
 'description_version': '1',
 'format': 'ARFF',
```

```
  'creator': ['Yann LeCun', 'Corinna Cortes', 'Christopher J.C. Burges'],
  'upload_date': '2014-09-29T03:28:38',
  'language': 'English',
  'licence': 'Public',
  'url': 'https://old.openml.org/data/v1/download/52667/mnist_784.arff',
  'file_id': '52667',
  'default_target_attribute': 'class',
  'tag': ['AzurePilot',
   'OpenML-CC18',
   'OpenML100',
   'study_1',
   'study_123',
   'study_41',
   'study_99',
   'vision'],
  'visibility': 'public',
  'minio_url': 'http://openml1.win.tue.nl/dataset554/dataset_554.pq',
  'status': 'active',
  'processing_date': '2020-11-20 20:12:09',
  'md5_checksum': '0298d579eb1b86163de7723944c7e495'},
 'url': 'https://www.openml.org/d/554'}
```

```
mnist.keys()
```

```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_names', 'DESCR'
```

```
X, y = mnist["data"], mnist["target"]
```

```
X.shape
```

```
(70000, 784)
```

```
y.shape
```

```
(70000,)
```

```
some_digit = X[0]
some_digit_image = some_digit.reshape(28,28)

plt.imshow(some_digit_image, cmap= "binary")
plt.axis("off")
plt.show()
```

```
y[0]
```

```
'5'
```

```
y = y.astype(np.uint8) #converts y[i]'s to integers
"""
Split the MNIST dataset into 60000 training images and 10000 for testing. Normally, we would
But instead we do PCA.
"""
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

We now conduct dimensionality reduction by principal component analysis (PCA). This reduces the time to run the OneVsRestClassifier/OneVsOneClassifier and even the SGD Classifier.

```
pca = PCA(n_components = 0.95)
X_pca = pca.fit_transform(X_train)
```

```
X_pca.shape #This only has 154 y-coordinates as opposed to 784
```

```
(60000, 154)
```

Let us see what dimensionality reduction has done to X[0] (we identified it earlier as 5). We use inverse_transform.

```
X_inv_pca = pca.inverse_transform(X_pca)
```

The X_inv_pca[0] is still 5 and dimensionality reduction has not affected it, as we see next.

```
same_digit = X_inv_pca[0]
same_digit_image = same_digit.reshape(28,28)

plt.imshow(same_digit_image, cmap= "binary")
plt.axis("off")
plt.show()
```

However, to make sure that PCA reduction has not distorted our image of 5 in X[0] significantly.

First we employ a support a support vector machine in conjunction with Principal Component Analysis dimensionality reduction.

```
svm =  SVC()
t_0 = time.time()
svm.fit(X_pca, y_train)
t_1 = time.time()

svm_training_time = t_1- t_0
print(svm_training_time)
```

```
111.74901819229126
```

```
svm.predict([X_pca[0]]) #does correctly identify it as 5
```

```
array([5], dtype=uint8)
```

```
cross_val_score(svm, X_pca, y_train, cv= 3, scoring = "accuracy") #it shows accuracy of rou
```

```
array([0.98045, 0.97745, 0.97885])
```

```
y_svm_pca = cross_val_predict(svm, X_pca, y_train, cv = 3)
print(y_svm_pca)
```

```
[5 0 4 ... 5 6 8]
```

```
svm_pca_precision = precision_score(y_train, y_svm_pca, average='weighted')
print(svm_pca_precision)
```

```
0.9789094200451905
```

```
svm_pca_recall = recall_score(y_train, y_svm_pca, average='weighted')
print(svm_pca_recall)
```

```
0.9789166666666667
```

We employ the OneVsRestClassifier.

```
ovr = OneVsRestClassifier(SVC())
t_3= time.time()
ovr.fit(X_pca, y_train)
t_4 = time.time()
```

```
ovr_training_time = t_4-t_3
print(ovr_training_time)
```

```
559.6759970188141
```

```
t_5 =time.time()
print(cross_val_score(ovr, X_pca, y_train, cv=3, scoring = "accuracy"))
t_6 = time.time()
```

```
[0.98185 0.97865 0.9787 ]
```

```
cross_val_ovr_training_time = t_6-t_5
print(cross_val_ovr_training_time)
```

```
866.3223266601562
```

```
y_ovr_pca = cross_val_predict(ovr, X_pca, y_train, cv = 3)
print(y_ovr_pca)
```

```
[5 0 4 ... 5 6 8]
```

```
ovr_pca_precision = precision_score(y_train, y_ovr_pca, average='weighted')
print(ovr_pca_precision)
```

```
0.9797318869328284
```

```
ovr_pca_recall = recall_score(y_train, y_ovr_pca, average='weighted')
print(ovr_pca_recall)
```

```
0.9797333333333333
```

The OneVsRestClassifier takes 648 seconds to train! We this resort to Incremental PCA, where we divide the MNIST dataset into 100 batches and use incremental PCA to reduce the dimension to 154 as before.

```
n_batches = 100
ipca = IncrementalPCA(n_components = 154)
t_7 = time.time()
for train_batch in np.array_split(X_train, n_batches):
    ipca.partial_fit(train_batch)

X_ipca = ipca.transform(X_train)
t_8 = time.time()

incremental_pca_training_time = t_8 - t_7
print(incremental_pca_training_time)
```

```
43.55912756919861
```

```
ovr_ipca = OneVsRestClassifier(SVC())

t_9= time.time()
ovr_ipca.fit(X_ipca, y_train)
t_10 = time.time()

ovr_ipca_training_time = t_10-t_9
print(t_10-t_9)
```

```
577.2291584014893
```

```
t_11 = time.time()
print(cross_val_score(ovr_ipca, X_ipca, y_train, cv = 3, scoring = "accuracy"))
t_12 = time.time()
```

```
[0.98195 0.97855 0.97885]
```

```
print(t_12-t_11) #cross validation time required for ovr_ipca is about 1058 seconds!
```

```
920.5151655673981
```

```
X_ipca.shape
```

```
(60000, 154)
```

```
y_ovr_ipca = cross_val_predict(ovr, X_ipca, y_train, cv = 3)
print(y_ovr_ipca)
```

```
[5 0 4 ... 5 6 8]
```

```
ovr_ipca_precision = precision_score(y_train, y_ovr_ipca, average='weighted')
print(ovr_ipca_precision)
```

```
0.9797836519174132
```

```
ovr_ipca_recall = recall_score(y_train, y_ovr_ipca, average='weighted')
print(ovr_ipca_recall)
```

```
0.9797833333333333
```

Let's give the Stochastic Gradient Descent a try.

```
t_13 = time.time()
sgd = SGDClassifier(random_state = 2)
sgd.fit(X_ipca, y_train)
t_14 = time.time()
```

```
sgd_time = t_14-t_13
print(sgd_time)
```

```
38.681522369384766
```

```
t_15 = time.time()
print(cross_val_score(sgd, X_ipca, y_train, cv = 3, scoring = "accuracy"))
t_16 = time.time()
```

```
[0.88265 0.8813  0.89215]
```

```
cross_validation_time_sgd = t_16-t_15
print(cross_validation_time_sgd)
```

```
76.45901226997375
```

```
y_sgd_ipca = cross_val_predict(sgd, X_ipca, y_train, cv = 3)
print(y_ovr_ipca)
```

```
[5 0 4 ... 5 6 8]
```

```
sgd_ipca_precision = precision_score(y_train, y_sgd_ipca, average='weighted')
print(ovr_ipca_precision)
```

```
0.9797836519174132
```

```
sgd_ipca_recall = recall_score(y_train, y_sgd_ipca, average='weighted')
print(ovr_ipca_recall)
```

```
0.9797833333333333
```

Just for the sake of completeness, let us now use the svm on X_ipac.

```
svm_ipca =  SVC()
t_17 = time.time()
svm_ipca.fit(X_ipca, y_train)
t_18 = time.time()
```

```
svm_ipca_train_time = t_18-t_17
print(svm_ipca_train_time)
```

```
110.42487001419067
```

```
t_19 = time.time()
print(cross_val_score(svm_ipca, X_ipca, y_train, cv = 3, scoring = "accuracy"))
t_20 = time.time()
```

```
[0.98045 0.9777  0.9791 ]
```

```
t_20-t_19 #time for cross validation cum_ipca
```

```
364.21145510673523
```

```
y_svm_ipca = cross_val_predict(svm, X_ipca, y_train, cv = 3)
print(y_svm_ipca)
```

```
[5 0 4 ... 5 6 8]
```

```
svm_ipca_precision = precision_score(y_train, y_svm_ipca, average='weighted')
print(svm_ipca_precision)
```

```
0.9790752567433006
```

```
svm_ipca_recall = recall_score(y_train, y_svm_ipca, average='weighted')
print(svm_ipca_recall)
```

```
0.9790833333333333
```

Random Forest Regressor

```
rf = RandomForestClassifier(n_estimators = 100)
t_21 = time.time()
rf.fit(X_ipca, y_train)
t_22 = time.time()
```

```
rf_train_time = t_22-t_21
print(rf_train_time)
```

```
120.94769597053528
```

```
t_23 = time.time()
print(cross_val_score(rf, X_ipca, y_train, cv= 3, scoring = "accuracy"))
t_24= time.time()
```

```
[0.94375 0.9372  0.94305]
```

```
t_24-t_23 #cross validation time for random forest
```

```
230.5682201385498
```

Let's just work with the Random Forest Classifier with just the X_pca

```
rf2 = RandomForestClassifier(n_estimators = 100)
t_25 = time.time()
rf2.fit(X_pca, y_train)
t_26=time.time()
```

```
rf2_train_time = t_26-t_25
print(rf2_train_time)
```

```
127.46941542625427
```

```
t_27 = time.time()
print(cross_val_score(rf, X_pca, y_train, cv= 3, scoring = "accuracy"))
t_28= time.time()
```

```
[0.9425 0.9381 0.9436]
```

```
t_28-t_27
```

```
231.1738612651825
```

And finally, just to satisfy our curiosity, we run a random forest classifier naively on X_train without dimensionality reduction

```
rf3 = RandomForestClassifier(n_estimators = 100)
t_29 = time.time()
rf3.fit(X_train, y_train)
t_30 = time.time()
```

```
rf3_train_time = t_30-t_29
print(rf3_train_time)
```

```
57.93721032142639
```

```
t_31 = time.time()
print(cross_val_score(rf3, X_train, y_train, cv= 3, scoring = "accuracy"))
t_32= time.time()
```

```
[0.9647 0.9633 0.9668]
```

```
t_32-t_31 #cross validation time for random forest without pca reduction on X_train
```

```
113.65859413146973
```

```
y_rf3 = cross_val_predict(rf3, X_train, y_train, cv=3)
print(y_rf3)
```

```
[5 0 4 ... 5 6 8]
```

```
rf3_precision = precision_score(y_train, y_rf3, average='weighted')
print(rf3_precision)
```

```
0.9652275266912358
```

```
rf3_recall = recall_score(y_train, y_rf3, average='weighted')
print(rf3_recall)
```

```
0.96525
```

We decide to go with the Support Vector Machine trained on X_pca which was simply dimension-reduced by principal component analysis, based on cross-validation score of roughly 98%, precision and recall of about 0.98 each, based on the training time of 111s. Other classifiers with similar cross-validation scores, precision and recall had greater training time.

We begin by dimensionality reduction on the X_test using PCA.

```
pca2 = PCA(n_components = 154)
X_pca_test = pca2.fit_transform(X_test)
```

```
X_pca_test.shape
```

```
(10000, 154)
```

```
y_predicted = svm.predict(X_pca_test)
```

```
incorrect_predictions = (y_test != y_predicted).sum()
print('The number of incorrect predictions is %d' %(y_test != y_predicted).sum())
```

```
The number of incorrect predictions is 8733
```

```
y_pred_rf3 = (y_test != (rf3.predict(X_test))).sum()
print(y_pred_rf3)
```

```
317
```

```
y_pred_ovr_pca = (y_test != (ovr.predict(X_pca_test))).sum()
print(y_pred_ovr_pca)
```

```
8737
```