ITMAL Exercise L07 - Gridsearch

Qa Explain GridSearchCV

There are two code cells below: 1:) function setup, 2) the actual grid-search.

Review the code cells and write a **short** summary. Mainly focus on **cell 2**, but dig into cell 1 if you find it interesting (notice the use of local-function, a nifty feature in python).

In detail, examine the lines:

```
grid_tuned = GridSearchCV(model, tuning_parameters, ..
grid_tuned.fit(X_train, y_train)
..
FullReport(grid_tuned , X_test, y_test, time_gridsearch)
```

and write a short description of how the `GridSeachCV` works: explain how the search parameter set is created and the overall search mechanism (without going into to much detail)

What role does the parameter `scoring='f1_micro'` play in the `GridSearchCV`, and what does `n_jobs=-1` mean?

In [1]:

```python
# TODO: Qa, code review..cell 1) function setup

from time import time
import numpy as np

from sklearn import svm
from sklearn.linear_model import SGDClassifier
from sklearn.neural_network import MLPClassifier

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, train_test_split
from sklearn.metrics import classification_report, f1_score
from sklearn import datasets

#from libitmal import utils as itmalutils
from libitmal import dataloaders_v2 as itmaldataloaders

currmode="N/A" # GLOBAL var!

def SearchReport(model):

    def GetBestModelCTOR(model, best_params):
        def GetParams(best_params):
            r=""
            for key in sorted(best_params):
                value = best_params[key]
                t = "'" if str(type(value))=="<class 'str'>" else ""
                if len(r)>0:
                    r += ','
                r += f'{key}={t}{value}{t}'
            return r
        try:
            p = GetParams(best_params)
            return type(model).__name__ + '(' + p + ')'
        except:
            return "N/A(1)"

    print("\nBest model set found on train set:")
    print()
    print(f"\tbest parameters={model.best_params_}")
    print(f"\tbest '{model.scoring}' score={model.best_score_}")
    print(f"\tbest index={model.best_index_}")
    print()
    print(f"Best estimator CTOR:")
    print(f"\t{model.best_estimator_}")
    print()
    try:
        print(f"Grid scores ('{model.scoring}') on development set:")
        means = model.cv_results_['mean_test_score']
        stds  = model.cv_results_['std_test_score']
        i=0
        for mean, std, params in zip(means, stds, model.cv_results_['params']):
            print("\t[%2d]: %0.3f (+/-%0.03f) for %r" % (i, mean, std * 2, params))
            i += 1
            if i == 20:
                break;
    except:
        print("WARNING: the random search do not provide means/stds")

    global currmode
    assert "f1_micro"==str(model.scoring), f"come on, we need to fix the scoring to be
 able to compare model-fits! Your scoreing={str(model.scoring)}...remember to add scori
ng='f1_micro' to the search"
    return f"best: dat={currmode}, score={model.best_score_:0.5f}, model={GetBestModelC
TOR(model.estimator,model.best_params_)}", model.best_estimator_

def ClassificationReport(model, X_test, y_test, target_names=None):
    assert X_test.shape[0]==y_test.shape[0]
    print("\nDetailed classification report:")
    print("\tThe model is trained on the full development set.")
    print("\tThe scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, model.predict(X_test)
    print(classification_report(y_true, y_pred, target_names))
    print()

def FullReport(model, X_test, y_test, t):
    print(f"SEARCH TIME: {t:0.2f} sec")
    beststr, bestmodel = SearchReport(model)
    ClassificationReport(model, X_test, y_test)
    print(f"CTOR for best model: {bestmodel}\n")
    print(f"{beststr}\n")
    return beststr, bestmodel

def LoadAndSetupData(mode, test_size=0.3):
    assert test_size>=0.0 and test_size<=1.0

    def ShapeToString(Z):
        n = Z.ndim
        s = "("
        for i in range(n):
            s += f"{Z.shape[i]:5d}"
            if i+1!=n:
                s += ";"
        return s+")"

    global currmode
    currmode=mode
    print(f"DATA: {currmode}..")

    if mode=='moon':
        X, y = itmaldataloaders.MOON_GetDataSet(n_samples=5000, noise=0.2)
        itmaldataloaders.MOON_Plot(X, y)
    elif mode=='mnist':
        X, y = itmaldataloaders.MNIST_GetDataSet(fetchmode=False)
        if X.ndim==3:
            X=np.reshape(X, (X.shape[0], -1))
    elif mode=='iris':
        X, y = itmaldataloaders.IRIS_GetDataSet()
    else:
        raise ValueError(f"could not load data for that particular mode='{mode}'")

    print(f'  org. data:  X.shape      ={ShapeToString(X)}, y.shape      ={ShapeToStrin
g(y)}')

    assert X.ndim==2
    assert X.shape[0]==y.shape[0]
    assert y.ndim==1 or (y.ndim==2 and y.shape[1]==0)

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=0, shuffle=True
    )

    print(f'  train data: X_train.shape={ShapeToString(X_train)}, y_train.shape={ShapeT
oString(y_train)}')
    print(f'  test data:  X_test.shape ={ShapeToString(X_test)}, y_test.shape ={ShapeTo
String(y_test)}')
    print()

    return X_train, X_test, y_train, y_test

print('OK')
```

```
OK
```

In [2]:

```
# TODO: Qa, code review..cell 2) the actual grid-search

# Setup data
X_train, X_test, y_train, y_test = LoadAndSetupData('iris') # or 'moon', or 'mnist'

# Setup search parameters
model = svm.SVC(gamma="scale")

# Setup tuning parameters
tuning_parameters = {
    'kernel':('linear', 'rbf'),
    'C':[1, 10]
}

# cross validation
CV=5

# Verbose logging
VERBOSE=0

# Run GridSearchCV for the model
start = time()
grid_tuned = GridSearchCV(model, tuning_parameters, cv=CV, scoring='f1_micro', verbose=
VERBOSE, n_jobs=-1, iid=True)

# The gridSearchCV function, is a way of

grid_tuned.fit(X_train, y_train)
t = time()-start

# Report result
b0, m0= FullReport(grid_tuned , X_test, y_test, t)
print('OK')
```

```
DATA: iris..
  org. data:  X.shape      =(  150;    4), y.shape      =(  150)
  train data: X_train.shape=(  105;    4), y_train.shape=(  105)
  test data:  X_test.shape =(   45;    4), y_test.shape =(   45)

SEARCH TIME: 2.65 sec

Best model set found on train set:

        best parameters={'C': 1, 'kernel': 'linear'}
        best 'f1_micro' score=0.9714285714285714
        best index=0

Best estimator CTOR:
        SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Grid scores ('f1_micro') on development set:
        [ 0]: 0.971 (+/-0.048) for {'C': 1, 'kernel': 'linear'}
        [ 1]: 0.952 (+/-0.084) for {'C': 1, 'kernel': 'rbf'}
        [ 2]: 0.952 (+/-0.084) for {'C': 10, 'kernel': 'linear'}
        [ 3]: 0.971 (+/-0.048) for {'C': 10, 'kernel': 'rbf'}

Detailed classification report:
        The model is trained on the full development set.
        The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.94      0.97        18
           2       0.92      1.00      0.96        11

   micro avg       0.98      0.98      0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45


CTOR for best model: SVC(C=1, cache_size=200, class_weight=None, coef0=0.
0,
  decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

best: dat=iris, score=0.97143, model=SVC(C=1,kernel='linear')

OK
```

Code summary:

The above code seeks to find the best fitted hyper parameters for the given model. In this example the support-vector classifier (SVC) is the one in play. The SVC model has a dozen of different parameteres (Hyper Parameters) which each can be tweaked and twisted in order for the model to better fit the data. To find the best suited combination of hyper parameters one would have to check each parameter against one another and calculate a score of how well the model is fit for the data.

Fortuanaly this process can be automatized using various methods. In this examples we use the GridsearchCV function. A grid search is basically brute forcing every possible combination of hyper parameters and finds the one with the highest score. To limit the amount of checks, one can pick which hyper parameters that have to be checked, in this case the parameters "kernel" and "C".

After the process is done the calculated best combinations on the iris data is with the hyperparameters of "kernel = linear" and "C = 1".

Qb Hyperparameter Grid Search using an SDG classifier

Now, replace the `svm.SVC` model with an `SGDClassifier` and a suitable set of the hyperparameters for that model.

You need at least four or five different hyperparameters from the `SDG` in the search-space before it begins to take considerable compute time doing the full grid search.

In [3]:

```
# TODO: Qb..
import warnings
warnings.filterwarnings('ignore')
# Setup search parameters
# model = svm.SVC(gamma="scale")
model = SGDClassifier() #alpha=0.01,verbose=0.1,eta0=0.1,power_t=0.0
# model = SGDClassifier(loss="hinge", penalty="l2", max_iter=5)

tuning_parameters = {
    'epsilon': [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001],
    'loss': ('log', 'hinge', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_
loss'),
    'penalty': ['l1', 'l2', 'elasticnet'],
    'alpha': [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001],
    'l1_ratio': [0.5, 0.1, 0.01, 0.001, 0.00001, 0.000001]
}

CV=5
VERBOSE=0

# Run GridSearchCV for the model
start = time()
grid_tuned = GridSearchCV(model, tuning_parameters, cv=CV, scoring='f1_micro', verbose=
VERBOSE, n_jobs=-1, iid=True)

# The gridSearchCV function, is a way of

grid_tuned.fit(X_train, y_train)
t = time()-start

# Report result
b0, m0= FullReport(grid_tuned , X_test, y_test, t)
print('OK')
```

```
SEARCH TIME: 24.03 sec

Best model set found on train set:

        best parameters={'alpha': 0.1, 'epsilon': 0.1, 'l1_ratio': 1e-05,
'loss': 'squared_hinge', 'penalty': 'l1'}
        best 'f1_micro' score=0.9904761904761905
        best index=81

Best estimator CTOR:
        SGDClassifier(alpha=0.1, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=1e-05, learning_rate='optimal', loss='squared_hinge',
       max_iter=None, n_iter=None, n_iter_no_change=5, n_jobs=None,
       penalty='l1', power_t=0.5, random_state=None, shuffle=True,
       tol=None, validation_fraction=0.1, verbose=0, warm_start=False)

Grid scores ('f1_micro') on development set:
        [ 0]: 0.800 (+/-0.133) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'log', 'penalty': 'l1'}
        [ 1]: 0.752 (+/-0.093) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'log', 'penalty': 'l2'}
        [ 2]: 0.686 (+/-0.091) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'log', 'penalty': 'elasticnet'}
        [ 3]: 0.790 (+/-0.203) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'hinge', 'penalty': 'l1'}
        [ 4]: 0.733 (+/-0.137) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'hinge', 'penalty': 'l2'}
        [ 5]: 0.762 (+/-0.183) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'hinge', 'penalty': 'elasticnet'}
        [ 6]: 0.943 (+/-0.072) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'modified_huber', 'penalty': 'l1'}
        [ 7]: 0.714 (+/-0.122) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'modified_huber', 'penalty': 'l2'}
        [ 8]: 0.943 (+/-0.141) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'modified_huber', 'penalty': 'elasticnet'}
        [ 9]: 0.867 (+/-0.232) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'squared_hinge', 'penalty': 'l1'}
        [10]: 0.667 (+/-0.370) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'squared_hinge', 'penalty': 'l2'}
        [11]: 0.886 (+/-0.211) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'squared_hinge', 'penalty': 'elasticnet'}
        [12]: 0.886 (+/-0.099) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'perceptron', 'penalty': 'l1'}
        [13]: 0.686 (+/-0.374) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'perceptron', 'penalty': 'l2'}
        [14]: 0.743 (+/-0.167) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'perceptron', 'penalty': 'elasticnet'}
        [15]: 0.343 (+/-0.079) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'squared_loss', 'penalty': 'l1'}
        [16]: 0.381 (+/-0.104) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'squared_loss', 'penalty': 'l2'}
        [17]: 0.295 (+/-0.279) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.5, 'loss': 'squared_loss', 'penalty': 'elasticnet'}
        [18]: 0.781 (+/-0.170) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.1, 'loss': 'log', 'penalty': 'l1'}
        [19]: 0.752 (+/-0.134) for {'alpha': 0.1, 'epsilon': 0.1, 'l1_rati
o': 0.1, 'loss': 'log', 'penalty': 'l2'}

Detailed classification report:
        The model is trained on the full development set.
        The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.89      0.94        18
           2       0.85      1.00      0.92        11

   micro avg       0.96      0.96      0.96        45
   macro avg       0.95      0.96      0.95        45
weighted avg       0.96      0.96      0.96        45


CTOR for best model: SGDClassifier(alpha=0.1, average=False, class_weight=
None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=1e-05, learning_rate='optimal', loss='squared_hinge',
       max_iter=None, n_iter=None, n_iter_no_change=5, n_jobs=None,
       penalty='l1', power_t=0.5, random_state=None, shuffle=True,
       tol=None, validation_fraction=0.1, verbose=0, warm_start=False)

best: dat=iris, score=0.99048, model=SGDClassifier(alpha=0.1,epsilon=0.1,l
1_ratio=1e-05,loss='squared_hinge',penalty='l1')

OK
```

Results

As seen in the code, four hyper parameters for the model is chosen (epsilon, loss, penalty, alpha). After doing a grid search, which resulted in over 4000 different combinations, the best model with a score of 0.99048 was found.

Best combination of chosen hyper parameters for the SGDClassifier model using iris data: (alpha=0.1,epsilon=0.1,l1_ratio=1e05,loss='squared_hinge',penalty='l1')

Qc Hyperparameter Random Search using an SDG classifier

Now, add code to run a `RandomizedSearchCV` instead.

Conceptual graphical view of randomized search for two distinct hyperparameters.

Use the same parameters for the random search, but add and investigate the new `n_iter` parameter

```
random_tuned = RandomizedSearchCV(model, tuning_parameters, random_state=42, n_i
ter=20, cv=CV, scoring='f1_micro', verbose=VERBOSE, n_jobs=-1, iid=True)
```

Comparison of time (seconds) to complete `GridSearch` versus `RandomizedSearchCV` , does not necessarily give any sense, if your grid search completes in seconds (for the iris tiny-data).

But you could compare the best-tuned parameter set and best scoring for the two methods. Is the random search best model close to the grid search?

In [4]:

```
# TODO: Qc..

# Run GridSearchCV for the model
start = time()
random_tuned = RandomizedSearchCV(model, tuning_parameters, random_state=42, n_iter=100
, cv=CV,
                                  scoring='f1_micro', verbose=VERBOSE, n_jobs=-1, iid=T
rue)
# The gridSearchCV function, is a way of

random_tuned.fit(X_train, y_train)
t = time()-start

# Report result
b0, m0= FullReport(random_tuned , X_test, y_test, t)

print('OK')
```

```
SEARCH TIME: 1.16 sec

Best model set found on train set:

        best parameters={'penalty': 'l1', 'loss': 'modified_huber', 'l1_ra
tio': 1e-05, 'epsilon': 0.0001, 'alpha': 0.1}
        best 'f1_micro' score=0.9619047619047619
        best index=29

Best estimator CTOR:
        SGDClassifier(alpha=0.1, average=False, class_weight=None,
       early_stopping=False, epsilon=0.0001, eta0=0.0, fit_intercept=True,
       l1_ratio=1e-05, learning_rate='optimal', loss='modified_huber',
       max_iter=None, n_iter=None, n_iter_no_change=5, n_jobs=None,
       penalty='l1', power_t=0.5, random_state=None, shuffle=True,
       tol=None, validation_fraction=0.1, verbose=0, warm_start=False)

Grid scores ('f1_micro') on development set:
        [ 0]: 0.733 (+/-0.225) for {'penalty': 'l1', 'loss': 'perceptron',
'l1_ratio': 1e-05, 'epsilon': 1e-06, 'alpha': 1e-06}
        [ 1]: 0.705 (+/-0.233) for {'penalty': 'elasticnet', 'loss': 'perc
eptron', 'l1_ratio': 0.01, 'epsilon': 0.0001, 'alpha': 0.0001}
        [ 2]: 0.324 (+/-0.051) for {'penalty': 'elasticnet', 'loss': 'squa
red_loss', 'l1_ratio': 0.01, 'epsilon': 1e-05, 'alpha': 0.0001}
        [ 3]: 0.695 (+/-0.251) for {'penalty': 'elasticnet', 'loss': 'squa
red_hinge', 'l1_ratio': 0.01, 'epsilon': 1e-06, 'alpha': 0.0001}
        [ 4]: 0.867 (+/-0.104) for {'penalty': 'l1', 'loss': 'squared_hing
e', 'l1_ratio': 1e-06, 'epsilon': 0.001, 'alpha': 0.1}
        [ 5]: 0.676 (+/-0.062) for {'penalty': 'l2', 'loss': 'hinge', 'l1_
ratio': 1e-05, 'epsilon': 0.0001, 'alpha': 1e-06}
        [ 6]: 0.933 (+/-0.138) for {'penalty': 'l1', 'loss': 'squared_hing
e', 'l1_ratio': 0.1, 'epsilon': 0.0001, 'alpha': 0.1}
        [ 7]: 0.648 (+/-0.059) for {'penalty': 'elasticnet', 'loss': 'hing
e', 'l1_ratio': 0.01, 'epsilon': 0.01, 'alpha': 0.1}
        [ 8]: 0.333 (+/-0.069) for {'penalty': 'l2', 'loss': 'squared_los
s', 'l1_ratio': 0.5, 'epsilon': 0.0001, 'alpha': 0.0001}
        [ 9]: 0.762 (+/-0.239) for {'penalty': 'l2', 'loss': 'modified_hub
er', 'l1_ratio': 0.1, 'epsilon': 1e-05, 'alpha': 0.1}
        [10]: 0.667 (+/-0.214) for {'penalty': 'l2', 'loss': 'log', 'l1_ra
tio': 0.1, 'epsilon': 1e-06, 'alpha': 1e-06}
        [11]: 0.762 (+/-0.239) for {'penalty': 'elasticnet', 'loss': 'hing
e', 'l1_ratio': 1e-06, 'epsilon': 0.01, 'alpha': 0.001}
        [12]: 0.657 (+/-0.395) for {'penalty': 'l2', 'loss': 'perceptron',
'l1_ratio': 0.1, 'epsilon': 0.0001, 'alpha': 0.001}
        [13]: 0.886 (+/-0.200) for {'penalty': 'l2', 'loss': 'squared_hing
e', 'l1_ratio': 0.1, 'epsilon': 1e-06, 'alpha': 0.1}
        [14]: 0.381 (+/-0.211) for {'penalty': 'elasticnet', 'loss': 'squa
red_loss', 'l1_ratio': 0.001, 'epsilon': 0.01, 'alpha': 0.1}
        [15]: 0.857 (+/-0.250) for {'penalty': 'elasticnet', 'loss': 'perc
eptron', 'l1_ratio': 0.5, 'epsilon': 0.1, 'alpha': 0.1}
        [16]: 0.248 (+/-0.273) for {'penalty': 'l1', 'loss': 'squared_los
s', 'l1_ratio': 0.1, 'epsilon': 1e-05, 'alpha': 0.01}
        [17]: 0.724 (+/-0.191) for {'penalty': 'l2', 'loss': 'modified_hub
er', 'l1_ratio': 1e-05, 'epsilon': 0.0001, 'alpha': 0.0001}
        [18]: 0.400 (+/-0.318) for {'penalty': 'elasticnet', 'loss': 'squa
red_loss', 'l1_ratio': 0.001, 'epsilon': 0.01, 'alpha': 1e-05}
        [19]: 0.686 (+/-0.337) for {'penalty': 'l1', 'loss': 'modified_hub
er', 'l1_ratio': 0.01, 'epsilon': 0.01, 'alpha': 0.001}

Detailed classification report:
        The model is trained on the full development set.
        The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.89      0.94        18
           2       0.85      1.00      0.92        11

   micro avg       0.96      0.96      0.96        45
   macro avg       0.95      0.96      0.95        45
weighted avg       0.96      0.96      0.96        45


CTOR for best model: SGDClassifier(alpha=0.1, average=False, class_weight=
None,
       early_stopping=False, epsilon=0.0001, eta0=0.0, fit_intercept=True,
       l1_ratio=1e-05, learning_rate='optimal', loss='modified_huber',
       max_iter=None, n_iter=None, n_iter_no_change=5, n_jobs=None,
       penalty='l1', power_t=0.5, random_state=None, shuffle=True,
       tol=None, validation_fraction=0.1, verbose=0, warm_start=False)

best: dat=iris, score=0.96190, model=SGDClassifier(alpha=0.1,epsilon=0.000
1,l1_ratio=1e-05,loss='modified_huber',penalty='l1')

OK
```

Result

This time instead of using the method grid search method a random search was used. This process is way lighter, but gives a more questionable result.

After running the code a couple of times, it seems that the score, using the random search, usually is about 0.05 below the grid search. However it takes around 0.15 seconds to execute whereas the grid search takes around 3 seconds. This is 20 times faster.

Qd Search Quest

Finally, we create a small competition: who can find the best model+hyperparameters for MNIST dataset?

You change to the MNIST data by calling `LoadAndSetupData('mnist')` , and this is a completely other ball-game that the *tiny-data* iris: it's much larger (but still far from *big-data*)!

- You might opt for an exhaustive grid search, or a faster but-less optimal random search...your choice.
- You are free to pick any classifier in Scikit-learn, even algorithms we have not discussed yet, but keep the score function at `f1_micro` , otherwise, we will be comparing 'æbler og pærer'.
- And, you may also want to scale you input data for some models to perform better (neural networks in particular).
- DO NOT USE Keras or Tensorflow models...not yet, and there are too many examples on the net to cut-and-paste from!

Check your result by printing the first *return* value from `FullReport()`

```
b1, m1 = FullReport(random_tuned , X_test, y_test, time_randomsearch)
print(b1)
```

that will display a result like

```
best: dat=iris, score=0.97143, model=SVC(C=1, kernel='linear')
```

Now, check if your score (for MNIST) is better that the currently best score on Blackboard: "L07: Optimization and searching" | "Search Quest for MNIST"

> https://blackboard.au.dk/webapps/blackboard/content/listContentEditable.jsp?content_id=_2117394_1&course_id=_124256_1&content_id=_2179138_1 (https://blackboard.au.dk/webapps/blackboard/content/listContentEditable.jsp?content_id=_2117394_1&course_id=_124256_1&content_id=_2179138_1)

and paste your best model into the message box, like

```
best(Mr.Itmal): dat=mnist, score=0.47090, model=MLPClassifier(random_state=42, m
ax_iter=10, activation='tanh')
```

Remember to provide a *custom* name manually, like 'best(joe)', 'best(john)' or 'best(it256)', so we can identify a winnner!

For the journal, report your progress in scoring choosing different models, hyperparameters to search and how you might need to preprocess your data...

In [5]:

```
# TODO: Qd..
# Setup data

#import importlib
#importlib.reload(itmaldataloaders)


from libitmal import kernelfuns as itmalkernelfuns
itmalkernelfuns.EnableGPU()

X_train, X_test, y_train, y_test = LoadAndSetupData('mnist')

model = MLPClassifier(max_iter=100)

tuning_parameters = {
    'hidden_layer_sizes': [(20, 50, 100, 100, 50, 20), (50,50,50)],
    'activation': ['tanh', 'relu'],
    'alpha': [0.0001, 0.000001, 0.05],
    'learning_rate' : ('constant', 'adaptive'),
    'solver' : ('sgd', 'adam')
}


start = time()
random_tuned = RandomizedSearchCV(model, tuning_parameters, n_iter=10, cv=CV, scoring=
'f1_micro', verbose=VERBOSE, n_jobs=-1, iid=True)
# The gridSearchCV function, is a way of

random_tuned.fit(X_train, y_train)

t = time()-start

b1, m1 = FullReport(random_tuned , X_test, y_test, t)

# best: dat=mnist, score=0.95933, model=MLPClassifier(activation='relu',alpha=0.05,hidd
en_layer_sizes=(50, 50, 50),learning_rate='adaptive',solver='adam')


#print(b1)
```

```
DATA: mnist..
  org. data:  X.shape      =(70000;  784), y.shape      =(70000)
  train data: X_train.shape=(49000;  784), y_train.shape=(49000)
  test data:  X_test.shape =(21000;  784), y_test.shape =(21000)

SEARCH TIME: 1612.69 sec

Best model set found on train set:

        best parameters={'solver': 'adam', 'learning_rate': 'adaptive', 'h
idden_layer_sizes': (50, 50, 50), 'alpha': 1e-06, 'activation': 'relu'}
        best 'f1_micro' score=0.9599387755102041
        best index=3

Best estimator CTOR:
        MLPClassifier(activation='relu', alpha=1e-06, batch_size='auto', b
eta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(50, 50, 50), learning_rate='adaptive',
       learning_rate_init=0.001, max_iter=100, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)

Grid scores ('f1_micro') on development set:
        [ 0]: 0.942 (+/-0.003) for {'solver': 'sgd', 'learning_rate': 'ada
ptive', 'hidden_layer_sizes': (50, 50, 50), 'alpha': 0.05, 'activation':
'tanh'}
        [ 1]: 0.953 (+/-0.007) for {'solver': 'adam', 'learning_rate': 'ad
aptive', 'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'alpha': 1e-06,
'activation': 'relu'}
        [ 2]: 0.930 (+/-0.011) for {'solver': 'adam', 'learning_rate': 'co
nstant', 'hidden_layer_sizes': (50, 50, 50), 'alpha': 0.05, 'activation':
'tanh'}
        [ 3]: 0.960 (+/-0.002) for {'solver': 'adam', 'learning_rate': 'ad
aptive', 'hidden_layer_sizes': (50, 50, 50), 'alpha': 1e-06, 'activation':
'relu'}
        [ 4]: 0.904 (+/-0.016) for {'solver': 'sgd', 'learning_rate': 'con
stant', 'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'alpha': 0.05,
'activation': 'tanh'}
        [ 5]: 0.928 (+/-0.010) for {'solver': 'adam', 'learning_rate': 'ad
aptive', 'hidden_layer_sizes': (50, 50, 50), 'alpha': 0.0001, 'activatio
n': 'tanh'}
        [ 6]: 0.929 (+/-0.010) for {'solver': 'adam', 'learning_rate': 'ad
aptive', 'hidden_layer_sizes': (50, 50, 50), 'alpha': 0.05, 'activation':
'tanh'}
        [ 7]: 0.931 (+/-0.014) for {'solver': 'sgd', 'learning_rate': 'con
stant', 'hidden_layer_sizes': (50, 50, 50), 'alpha': 1e-06, 'activation':
'tanh'}
        [ 8]: 0.906 (+/-0.012) for {'solver': 'adam', 'learning_rate': 'co
nstant', 'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'alpha': 0.000
1, 'activation': 'tanh'}
        [ 9]: 0.960 (+/-0.009) for {'solver': 'adam', 'learning_rate': 'co
nstant', 'hidden_layer_sizes': (50, 50, 50), 'alpha': 0.05, 'activation':
'relu'}

Detailed classification report:
        The model is trained on the full development set.
        The scores are computed on the full evaluation set.

              precision    recall  f1-score   support
```

```
           0       0.99      0.97      0.98      2077
           1       0.98      0.99      0.99      2385
           2       0.95      0.97      0.96      2115
           3       0.95      0.96      0.96      2117
           4       0.97      0.95      0.96      2004
           5       0.96      0.95      0.96      1900
           6       0.97      0.97      0.97      2045
           7       0.97      0.96      0.97      2189
           8       0.95      0.95      0.95      2042
           9       0.93      0.96      0.94      2126

   micro avg       0.96      0.96      0.96     21000
   macro avg       0.96      0.96      0.96     21000
weighted avg       0.96      0.96      0.96     21000


CTOR for best model: MLPClassifier(activation='relu', alpha=1e-06, batch_s
ize='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(50, 50, 50), learning_rate='adaptive',
       learning_rate_init=0.001, max_iter=100, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)

best: dat=mnist, score=0.95994, model=MLPClassifier(activation='relu',alph
a=1e-06,hidden_layer_sizes=(50, 50, 50),learning_rate='adaptive',solver='a
dam')
```

Results

This exercise was much like the previous, however instead of re-using the model from the other exercises we chose a new model (MLPClassifier) which supports neural networks. By using neural networks one can achieve a significantly more precise model, however the training becomes equally heavy (to the point were a single process took over 12 minutes).

Due to the heaviness of the process and the amount of data in mnist, not a lot of combinations were tested. The best found combination was:

MLPClassifier(activation='relu',alpha=1e-06,hidden_layer_sizes=(50, 50, 50),learning_rate='adaptive',solver='adam') with a score of 0.959943