Supervised Algorithms Comparison

Amir Rashidi
University of California San Diego
arashidi@ucsed.edu

ABSTRACT

This paper compares SVM, Random Forests, and Logistic Regression performance on multiple data sets from UCI data base [1].

1 INTRODUCTION

In this paper, we will compare the accuracy and efficiency of some of the most common supervised algorithms and try to analyze their behavior using our data sets.
The goal of this paper is to gain a better understanding of how these algorithms would perform on multiple data sets.
The data sets are:

1.*ISOLET* Which is encoded voice [1]

2.*LetterRecognition* Which has images of letters and the algorithm's task is to recognize the image [1]

3.*Dataset for Sensorless Drive Diagnosis* (will be referred to as *sens* for the rest of the paper) "Features are extracted from electric current drive signals. The drive has intact and defective components. This results in 11 different classes with different conditions. Each condition has been measured several times by 12 different operating conditions, this means by different speeds, load moments and load forces. The current signals are measured with a current probe and an oscilloscope on two phases.".[1]

Data Information			
Dataset Name	*NumberofAttributes*	Train Size	Test Size
Isolet	617	5000	1238
Sens	48	5000	53508
Letters	16	5000	14999

2 METHODS

The code used for these calculations is going to use 5000 training points from each data set, and it'll use the remaining for testing.

For selecting our hyperparameters(HP), we'll be trying multiple HPs and select the one with the best outcome to be used for testing. We use *GridSearchCV*() to get the best HPs for our classifiers.

2.1 SVM

For SVM we use a commonly known library from *sklearn* also known as *SVC*. We'll try *rbf* kernel, *linear* Kernel, and *sigmoid* Kernel. We'll compare their results to find the most optimal classifier for our data sets. For our C, we'll use 10^{-3} to 10^{3} and for *Gamma* of *rbf* we used 10^{-5} to 1

2.2 Random Forests

We'll be using *sklearn* function called *RandomForestClassifier*() and we'll use estimators 10, 100, 1000 and $max_features$ of 1, 2, 3

2.3 Logistic regression

We'll be using *sklearn* library function called *LogisticRegression*(). For our C, we'll use 10^-4 to 10^4 and for our classifier penalty, we'll use $l1$ and $l2$.

3 EXPERIMENTS

For our experiment, we used the first 5000 data points of every data sets for training purposes and validation, and the rest of the data set was used for testing purposes.

3.1 cross validation

For our cross validation, we used Mean Squared Error as our scorers with $cv = 5$ (Cross Validation folds).

3.2 Learning Curve

We applied learning curve to see how the data is behaving and we used max_error for our error evaluation.

3.3 Formulas

The formula for mean absolute error(MAE) [2] is:

$$MAE(y, y') = 1/n_{samples} \sum_{i=0}^{n_{samples}-1} |y_i - y'_i|$$

and the formula for max error which is also known as residual error [2] is

$$MaxError(y, y') = max(|y_i - y'_i|)$$

4 CONCLUSION/RESULTS

4.1 table of results

Results			
Metric/Aspect	*Random Forest*	*Support Vector Machine*	*LogisticRegression*
Best Parameters	criterion=gini,classifier n estimators=1000, classifier max features=2	'classifier C': 10.0, 'classifier kernel'= 'rbf', 'classifier gamma'= 0.01	'classifier penalty'= 'l2', 'classifier C'= 10.0
Outer CV accuracy mean	95.06% + 3.379	96.64% + 2.335	89.99% + 9.266
Letters			
Inner CV accuracy mean	89.58%	91.30%	74.44%
Training Accuracy	100.00%	99.92%	78.66%
Testing Accuracy	92.89%	94.47%	76.80%
F1 score	0.93%	0.94	0.77
Average split0 Test Score	0.85 +|- 0.00383%	0.4% +|- 0.00337	64% +|- 0.0075
Average split0 Train Score	1.00%	0.43 +|- 0.00252	0.67% +|- 0.0021 74
ISOLET			
Inner CV accuracy mean	92.80%	95.90%	95.14%
Training Accuracy	100.00%	100.00%	100.00%
Testing Accuracy	93.70%	96.53%	94.91%
F1 Testing score	0.94	0.97	0.95
Average split0 Test Score	0.83 +|- 0.00535%	0.72% +|- 0.0724	93% +|- 0.00189
Average split0 Train Score	1.00	0.93 +|- 0.1087	0.98% +|- 0.0001
Sensorless Drive Diagnosis			
Inner CV accuracy mean	99.84%	99.4 %	96.92 %
Training Accuracy	100.00%	99.9%	97.82%
Testing Accuracy	99.86%	99.33%	97.41%
F1 Testing score	1.00	0.99	0.97
Average split0 Test Score	0.99	0.83% +|- 0.01726%	90% +|- 0.00387
Average split0 Train Score	1	0.85% +|- 0.01477%	0.92% +|- 0.0067
Mean Results across data sets			
Inner CV accuracy mean	94.074%	95.53%	88.9%
Training Accuracy	100.00%	99.9%	92.16%
Testing Accuracy	95.48%	96.77%	89.7%
F1 Testing score	0.956	0.97	0.896
Average split0 Test Score	0.89	0.65	0.82
Average split0 Train Score	1	0.73	0.86

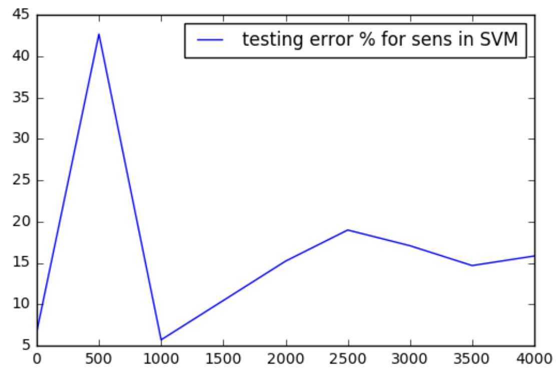
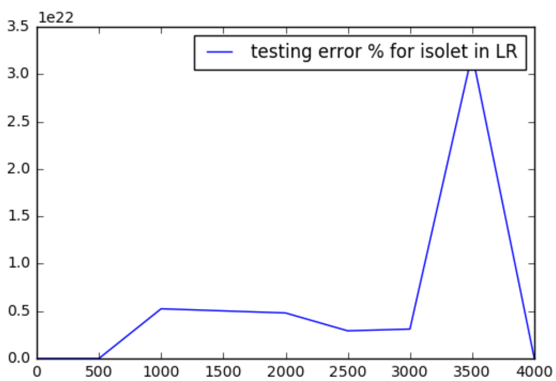


Figure 1: This data was obtained by applying [1,500,1000,2000,2500,3000,3500,4000] training sizes and using negative mean squared to calculate the error.



4.2 Analysis and Conclusion

The overall take from this paper is that these methods should be applied to the data sets which they are more applicable to. If there are data sets where the number of attributes are huge (Such as ISOLET data set), the SVM algorithm will have a hard time coming up with planes for classification in higher dimension between larger number of attributes but RF will be able to fit it a lot faster with a few percentage less accuracy on test sets. For some of this work, the data needed to become normalized and there was still room for improvement across the paper. Overall from our current result the Random Forrest seems to have the optimal results for testing and training overall in our three current data sets.
Another issue that this paper could look into in the future is the problem of over-fitting with Random Forest algorithm which could be managed. The graphs were produced via learning curve [2] with negative mean squared error calculation that has the formula mentioned in part 2.
The hope is to continue completing this paper during my leisure time to have a solid reference for myself and others for ML algorithms and selecting the right Classifier for the problem.

5 CODE AND EXTRA DATA TO REPORT

REFERENCES

[1] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

In [9]:
```
import numpy as np
import pandas as pd
import matplotlib as plt
import sklearn
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import validation_curve
from sklearn.metrics import r2_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import learning_curve
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt

# import seaborn as sns; sns.set_style('white')  # plot formatting

import warnings
# there are a lot of convergence warnings for some params, however be careful
 with this!!
# sometimes you need to see those wanrings, and now we've screwed tha tup for
 the whole notebook from here on!!
warnings.filterwarnings('ignore')
```

This is text

In [10]:
```
isolet_tuple = pd.read_csv("./ISOLET/isolet1+2+3+4.data", header=None, delim_w
hitespace=False)
isolet = pd.DataFrame(isolet_tuple)

isolet_features = isolet.drop(isolet.columns[[len(isolet.iloc[0]) - 1]], axis=
1)
isolet_labels = isolet.iloc[:, -1:]


letters_tuple = pd.read_csv("./Letter Recognition/letter-recognition.data")
letters_tuple.columns = ["letter", "x-box", "y-box", "width", "high", "onpix",
"x-bar", "y-bar", "x2bar", "y2bar",
                         "xybar", "x2ybr", "xy2br", "x-ege", "xegvy", "y-ege",
"yegvx"]
letters = pd.DataFrame(letters_tuple)
letters_features = letters.drop(letters.columns[[0]], axis=1)
letters_label = letters.iloc[:, 0]

sens_less = pd.read_csv("./Sensorless Drive Diagnosis/Sensorless_drive_diagnos
is.txt", sep=" ")
sens_features = sens_less.drop(sens_less.columns[[len(sens_less.iloc[0]) - 1
]], axis=1)
sens_labels = sens_less.iloc[:, -1:]
```

In [3]:
```
# isolet_tuple
```

In [21]:
```
from sklearn.model_selection import RepeatedKFold
clfRF = RandomForestClassifier()

clfSVM = SVC(random_state=12345)

clfReg = LogisticRegression(multi_class='multinomial',
                          solver='newton-cg',
                          random_state=12345)
pipe1 = Pipeline([('std', StandardScaler()),
                 ('classifier', clfRF)])

pipe2 = Pipeline([('std', StandardScaler()),
                  ('classifier', clfSVM)])

pipe3 = Pipeline([('std', StandardScaler()),
                  ('classifier', clfReg)])

# Create search space of candidate learning algorithms and their hyperparamete
rs
param_grid_RF = [{'classifier': [RandomForestClassifier()],
                 'classifier__n_estimators': [10, 100, 1000],
                 'classifier__max_features': [1, 2, 3]}]
param_grid_svm = [{'classifier__kernel': ['rbf'],
                'classifier__C': np.power(10., np.arange(-3, 3)),
                'classifier__gamma': np.power(10., np.arange(-5, 0))},
               {'classifier__kernel': ['linear'],
                'classifier__C': np.power(10., np.arange(-3, 3))},
               {'classifier__kernel': ['sigmoid'],
                'classifier__C': np.power(10., np.arange(-3, 3))},
                  ]

param_grid_logistic = [{'classifier__penalty': ['l2'],
                       'classifier__C': np.power(10., np.arange(-4, 4))}]
```

In [12]:
```
griddles = {} #Yummy
#isolet_train_x, isolet_test_x, isolet_train_y, isolet_test_y = train_test_spl
it(isolet_features[:150], isolet_labels[:150], test_size=0.2, random_state=30)
isolet_train_x, isolet_test_x, isolet_train_y, isolet_test_y = train_test_spli
t(isolet_features, isolet_labels, train_size=5000, random_state=12345, stratif
y=isolet_labels)
letters_train_x, letters_test_x, letters_train_y, letters_test_y = train_test_
split(letters_features[:int(len(letters_features))], letters_label[:int(len(le
tters_features))], train_size=5000, random_state=12345, stratify=letters_label
[:int(len(letters_features))])
sens_train_x, sens_test_x, sens_train_y, sens_test_y = train_test_split(sens_f
eatures[:int(len(sens_features)/2)], sens_labels[:int(len(sens_features)/2)],t
rain_size=5000, random_state=12345, stratify=sens_labels[:int(len(sens_feature
s)/2)])
```

In [13]:
```
data = {
    'Dataset' : ['ISOLET', 'SENS', 'Letters'],
    'Number of Attributes' : [len(isolet_features.iloc[0]), len(sens_features.
iloc[0]), len(letters_features.iloc[0])],
    'Train Size' : [5000,5000,5000],
    'Test Size' : [len(isolet_features)- 5000, len(sens_features)-5000, len(le
tters_features)-5000]
}

data_desc = pd.DataFrame(data)
# accur = {'Inner Accuracy', 'Outer Accuracy'}
acc_df = pd.DataFrame(columns=['name', 'dataset', 'outer', 'inner'])
# acc_df.append({'outer':500,'inner':200}, ignore_index=True)
```

In [14]:
```
%%time
for param_g, estimates, names in zip((param_grid_RF, param_grid_svm, param_gri
d_logistic),(pipe1, pipe2, pipe3),('RF','SVM', 'LR')):
    gc = GridSearchCV(estimator=estimates, param_grid=param_g, scoring='accura
cy', n_jobs=1, cv=2, verbose=0, refit=True)
    griddles[names] = gc
```

```
CPU times: user 54 µs, sys: 5 µs, total: 59 µs
Wall time: 60.6 µs
```

In [18]:
```
%%time
cv_scores = {name: [] for name, gs_est in griddles.items()}
#clf = GridSearchCV(pipe, search_space, cv=StratifiedKFold(n_splits=10), verbo
se=0)
# skfolded = StratifiedKFold(n_splits=2, shuffle=True, random_state=1)
#best_model = clf.fit(isolet_train_x, isolet_train_y)
skfolded= RepeatedKFold(n_splits=5, n_repeats=3, random_state=12345)
c=1
```

```
CPU times: user 26 µs, sys: 0 ns, total: 26 µs
Wall time: 29.6 µs
```

In [23]:
```
%%time
for i,j,k in ([ (letters_train_x, letters_train_y, 'letters'),(isolet_train_x,
isolet_train_y, 'isolet'), (sens_train_x, sens_train_y, 'sens')]):
    for outer_tr_ind, outer_val_ind in skfolded.split(i, j):
        print('_________________________________________________________')
        %%time
        for name, gs_est in sorted(griddles.items()):
            #print(j)
            print('dataset:%-8s outer fold %d/5 | tuning %-8s' % (k, c, name),
end='')
            #print(isolet_train_x.iloc[outer_tr_ind])
            gs_est.fit(i.iloc[outer_tr_ind], j.iloc[outer_tr_ind])
            y_pred = gs_est.predict(i.iloc[outer_val_ind])
            acc = accuracy_score(y_true=j.iloc[outer_val_ind], y_pred=y_pred)
            acc_df.append({'name': name, 'dataset': k, 'inner':gs_est.best_sco
re_ *100, 'outter':acc*100}, ignore_index=True)
            print(' | inner Accuracy %.2f%% | outer Accuracy %.2f%%' % (gs_es
t.best_score_ * 100, acc * 100))
            cv_scores[name].append(acc)
        c+=1
    c = 1
#best_model.best_estimator_.get_params()['classifier']
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 3.81 µs
dataset:letters  outer fold 1/5 | tuning LR      | inner Accuracy 74.65% | o
uter Accuracy 75.10%
dataset:letters  outer fold 1/5 | tuning RF      | inner Accuracy 87.55% | o
uter Accuracy 92.90%
dataset:letters  outer fold 1/5 | tuning SVM     | inner Accuracy 89.55% | o
uter Accuracy 93.90%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 4.77 µs
dataset:letters  outer fold 2/5 | tuning LR      | inner Accuracy 74.78% | o
uter Accuracy 76.90%
dataset:letters  outer fold 2/5 | tuning RF      | inner Accuracy 88.33% | o
uter Accuracy 91.60%
dataset:letters  outer fold 2/5 | tuning SVM     | inner Accuracy 89.68% | o
uter Accuracy 92.70%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 4.29 µs
dataset:letters  outer fold 3/5 | tuning LR      | inner Accuracy 74.62% | o
uter Accuracy 76.40%
dataset:letters  outer fold 3/5 | tuning RF      | inner Accuracy 88.30% | o
uter Accuracy 92.40%
dataset:letters  outer fold 3/5 | tuning SVM     | inner Accuracy 90.12% | o
uter Accuracy 93.30%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
dataset:letters  outer fold 4/5 | tuning LR      | inner Accuracy 74.12% | o
uter Accuracy 77.80%
dataset:letters  outer fold 4/5 | tuning RF      | inner Accuracy 87.95% | o
uter Accuracy 91.20%
dataset:letters  outer fold 4/5 | tuning SVM     | inner Accuracy 89.50% | o
uter Accuracy 93.80%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.05 µs
dataset:letters  outer fold 5/5 | tuning LR      | inner Accuracy 75.10% | o
uter Accuracy 76.30%
dataset:letters  outer fold 5/5 | tuning RF      | inner Accuracy 87.90% | o
uter Accuracy 93.00%
dataset:letters  outer fold 5/5 | tuning SVM     | inner Accuracy 89.50% | o
uter Accuracy 94.10%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.77 µs
dataset:letters  outer fold 6/5 | tuning LR      | inner Accuracy 74.60% | o
uter Accuracy 77.30%
dataset:letters  outer fold 6/5 | tuning RF      | inner Accuracy 87.67% | o
uter Accuracy 92.20%
dataset:letters  outer fold 6/5 | tuning SVM     | inner Accuracy 89.48% | o
uter Accuracy 95.30%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.29 µs
```

```
dataset:letters  outer fold 7/5 | tuning LR      | inner Accuracy 75.17% | o
uter Accuracy 77.80%
dataset:letters  outer fold 7/5 | tuning RF      | inner Accuracy 87.78% | o
uter Accuracy 92.70%
dataset:letters  outer fold 7/5 | tuning SVM     | inner Accuracy 89.75% | o
uter Accuracy 93.70%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
dataset:letters  outer fold 8/5 | tuning LR      | inner Accuracy 75.15% | o
uter Accuracy 77.00%
dataset:letters  outer fold 8/5 | tuning RF      | inner Accuracy 88.70% | o
uter Accuracy 93.00%
dataset:letters  outer fold 8/5 | tuning SVM     | inner Accuracy 89.48% | o
uter Accuracy 94.10%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 4.53 µs
dataset:letters  outer fold 9/5 | tuning LR      | inner Accuracy 75.00% | o
uter Accuracy 75.30%
dataset:letters  outer fold 9/5 | tuning RF      | inner Accuracy 88.78% | o
uter Accuracy 91.70%
dataset:letters  outer fold 9/5 | tuning SVM     | inner Accuracy 89.92% | o
uter Accuracy 93.20%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 4.53 µs
dataset:letters  outer fold 10/5 | tuning LR      | inner Accuracy 74.10% |
outer Accuracy 74.40%
dataset:letters  outer fold 10/5 | tuning RF      | inner Accuracy 87.90% |
outer Accuracy 92.20%
dataset:letters  outer fold 10/5 | tuning SVM     | inner Accuracy 89.42% |
outer Accuracy 94.00%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.29 µs
dataset:letters  outer fold 11/5 | tuning LR      | inner Accuracy 75.28% |
outer Accuracy 77.70%
dataset:letters  outer fold 11/5 | tuning RF      | inner Accuracy 88.08% |
outer Accuracy 92.30%
dataset:letters  outer fold 11/5 | tuning SVM     | inner Accuracy 89.00% |
outer Accuracy 94.20%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.01 µs
dataset:letters  outer fold 12/5 | tuning LR      | inner Accuracy 74.45% |
outer Accuracy 76.30%
dataset:letters  outer fold 12/5 | tuning RF      | inner Accuracy 88.15% |
outer Accuracy 91.90%
dataset:letters  outer fold 12/5 | tuning SVM     | inner Accuracy 89.55% |
outer Accuracy 93.90%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
dataset:letters  outer fold 13/5 | tuning LR      | inner Accuracy 75.02% |
outer Accuracy 76.90%
dataset:letters  outer fold 13/5 | tuning RF      | inner Accuracy 88.10% |
```

```
outer Accuracy 91.30%
dataset:letters  outer fold 13/5 | tuning SVM      | inner Accuracy 89.65% |
outer Accuracy 93.30%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.77 µs
dataset:letters  outer fold 14/5 | tuning LR       | inner Accuracy 74.33% |
outer Accuracy 76.70%
dataset:letters  outer fold 14/5 | tuning RF       | inner Accuracy 87.60% |
outer Accuracy 92.30%
dataset:letters  outer fold 14/5 | tuning SVM      | inner Accuracy 89.00% |
outer Accuracy 94.40%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.77 µs
dataset:letters  outer fold 15/5 | tuning LR       | inner Accuracy 75.52% |
outer Accuracy 75.90%
dataset:letters  outer fold 15/5 | tuning RF       | inner Accuracy 88.08% |
outer Accuracy 91.30%
dataset:letters  outer fold 15/5 | tuning SVM      | inner Accuracy 89.98% |
outer Accuracy 92.90%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.29 µs
dataset:isolet   outer fold 1/5 | tuning LR       | inner Accuracy 94.40% | o
uter Accuracy 95.90%
dataset:isolet   outer fold 1/5 | tuning RF       | inner Accuracy 92.75% | o
uter Accuracy 93.20%
dataset:isolet   outer fold 1/5 | tuning SVM      | inner Accuracy 95.43% | o
uter Accuracy 97.30%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
dataset:isolet   outer fold 2/5 | tuning LR       | inner Accuracy 94.58% | o
uter Accuracy 95.90%
dataset:isolet   outer fold 2/5 | tuning RF       | inner Accuracy 91.72% | o
uter Accuracy 94.00%
dataset:isolet   outer fold 2/5 | tuning SVM      | inner Accuracy 95.33% | o
uter Accuracy 97.60%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
dataset:isolet   outer fold 3/5 | tuning LR       | inner Accuracy 94.58% | o
uter Accuracy 95.00%
dataset:isolet   outer fold 3/5 | tuning RF       | inner Accuracy 91.88% | o
uter Accuracy 94.50%
dataset:isolet   outer fold 3/5 | tuning SVM      | inner Accuracy 95.67% | o
uter Accuracy 96.60%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.29 µs
dataset:isolet   outer fold 4/5 | tuning LR       | inner Accuracy 94.88% | o
uter Accuracy 94.70%
dataset:isolet   outer fold 4/5 | tuning RF       | inner Accuracy 92.25% | o
uter Accuracy 91.80%
dataset:isolet   outer fold 4/5 | tuning SVM      | inner Accuracy 95.35% | o
uter Accuracy 95.50%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.77 µs
dataset:isolet   outer fold 5/5 | tuning LR       | inner Accuracy 94.65% | o
uter Accuracy 96.30%
dataset:isolet   outer fold 5/5 | tuning RF       | inner Accuracy 91.90% | o
uter Accuracy 93.50%
dataset:isolet   outer fold 5/5 | tuning SVM      | inner Accuracy 95.17% | o
uter Accuracy 97.30%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 4.53 µs
dataset:isolet   outer fold 6/5 | tuning LR       | inner Accuracy 94.75% | o
uter Accuracy 95.50%
dataset:isolet   outer fold 6/5 | tuning RF       | inner Accuracy 92.33% | o
uter Accuracy 93.70%
dataset:isolet   outer fold 6/5 | tuning SVM      | inner Accuracy 95.33% | o
uter Accuracy 96.50%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.77 µs
dataset:isolet   outer fold 7/5 | tuning LR       | inner Accuracy 94.20% | o
uter Accuracy 96.40%
dataset:isolet   outer fold 7/5 | tuning RF       | inner Accuracy 92.35% | o
uter Accuracy 94.00%
dataset:isolet   outer fold 7/5 | tuning SVM      | inner Accuracy 95.00% | o
uter Accuracy 97.20%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 4.77 µs
dataset:isolet   outer fold 8/5 | tuning LR       | inner Accuracy 94.58% | o
uter Accuracy 95.90%
dataset:isolet   outer fold 8/5 | tuning RF       | inner Accuracy 92.20% | o
uter Accuracy 93.10%
dataset:isolet   outer fold 8/5 | tuning SVM      | inner Accuracy 95.38% | o
uter Accuracy 96.70%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
dataset:isolet   outer fold 9/5 | tuning LR       | inner Accuracy 94.92% | o
uter Accuracy 95.70%
dataset:isolet   outer fold 9/5 | tuning RF       | inner Accuracy 92.55% | o
uter Accuracy 92.30%
dataset:isolet   outer fold 9/5 | tuning SVM      | inner Accuracy 95.58% | o
uter Accuracy 97.10%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 4.29 µs
dataset:isolet   outer fold 10/5 | tuning LR       | inner Accuracy 94.70% |
outer Accuracy 95.20%
dataset:isolet   outer fold 10/5 | tuning RF       | inner Accuracy 92.05% |
outer Accuracy 94.20%
dataset:isolet   outer fold 10/5 | tuning SVM      | inner Accuracy 95.88% |
outer Accuracy 96.90%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 5.01 µs
```

```
dataset:isolet   outer fold 11/5 | tuning LR       | inner Accuracy 94.17% |
outer Accuracy 96.40%
dataset:isolet   outer fold 11/5 | tuning RF       | inner Accuracy 91.95% |
outer Accuracy 93.50%
dataset:isolet   outer fold 11/5 | tuning SVM      | inner Accuracy 95.17% |
outer Accuracy 97.70%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
dataset:isolet   outer fold 12/5 | tuning LR       | inner Accuracy 94.73% |
outer Accuracy 95.50%
dataset:isolet   outer fold 12/5 | tuning RF       | inner Accuracy 92.70% |
outer Accuracy 92.60%
dataset:isolet   outer fold 12/5 | tuning SVM      | inner Accuracy 95.25% |
outer Accuracy 96.60%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.29 µs
dataset:isolet   outer fold 13/5 | tuning LR       | inner Accuracy 94.75% |
outer Accuracy 95.80%
dataset:isolet   outer fold 13/5 | tuning RF       | inner Accuracy 92.00% |
outer Accuracy 93.00%
dataset:isolet   outer fold 13/5 | tuning SVM      | inner Accuracy 95.35% |
outer Accuracy 96.50%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.29 µs
dataset:isolet   outer fold 14/5 | tuning LR       | inner Accuracy 94.95% |
outer Accuracy 94.50%
dataset:isolet   outer fold 14/5 | tuning RF       | inner Accuracy 92.10% |
outer Accuracy 93.30%
dataset:isolet   outer fold 14/5 | tuning SVM      | inner Accuracy 95.65% |
outer Accuracy 95.70%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 5.25 µs
dataset:isolet   outer fold 15/5 | tuning LR       | inner Accuracy 95.25% |
outer Accuracy 95.80%
dataset:isolet   outer fold 15/5 | tuning RF       | inner Accuracy 92.55% |
outer Accuracy 94.70%
dataset:isolet   outer fold 15/5 | tuning SVM      | inner Accuracy 95.60% |
outer Accuracy 96.60%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 5.72 µs
dataset:sens     outer fold 1/5 | tuning LR       | inner Accuracy 96.62% | o
uter Accuracy 97.60%
dataset:sens     outer fold 1/5 | tuning RF       | inner Accuracy 99.80% | o
uter Accuracy 99.60%
dataset:sens     outer fold 1/5 | tuning SVM      | inner Accuracy 99.12% | o
uter Accuracy 99.50%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.05 µs
dataset:sens     outer fold 2/5 | tuning LR       | inner Accuracy 96.95% | o
uter Accuracy 97.20%
dataset:sens     outer fold 2/5 | tuning RF       | inner Accuracy 99.75% | o
```

```
uter Accuracy 100.00%
dataset:sens     outer fold 2/5 | tuning SVM      | inner Accuracy 99.30% | o
uter Accuracy 99.10%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
dataset:sens     outer fold 3/5 | tuning LR       | inner Accuracy 97.10% | o
uter Accuracy 96.80%
dataset:sens     outer fold 3/5 | tuning RF       | inner Accuracy 99.83% | o
uter Accuracy 99.80%
dataset:sens     outer fold 3/5 | tuning SVM      | inner Accuracy 99.17% | o
uter Accuracy 99.10%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.29 µs
dataset:sens     outer fold 4/5 | tuning LR       | inner Accuracy 96.40% | o
uter Accuracy 97.20%
dataset:sens     outer fold 4/5 | tuning RF       | inner Accuracy 99.80% | o
uter Accuracy 100.00%
dataset:sens     outer fold 4/5 | tuning SVM      | inner Accuracy 99.25% | o
uter Accuracy 99.50%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.05 µs
dataset:sens     outer fold 5/5 | tuning LR       | inner Accuracy 96.92% | o
uter Accuracy 96.60%
dataset:sens     outer fold 5/5 | tuning RF       | inner Accuracy 99.80% | o
uter Accuracy 100.00%
dataset:sens     outer fold 5/5 | tuning SVM      | inner Accuracy 99.22% | o
uter Accuracy 99.40%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
dataset:sens     outer fold 6/5 | tuning LR       | inner Accuracy 96.70% | o
uter Accuracy 97.70%
dataset:sens     outer fold 6/5 | tuning RF       | inner Accuracy 99.78% | o
uter Accuracy 99.90%
dataset:sens     outer fold 6/5 | tuning SVM      | inner Accuracy 99.17% | o
uter Accuracy 99.30%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
dataset:sens     outer fold 7/5 | tuning LR       | inner Accuracy 96.40% | o
uter Accuracy 97.40%
dataset:sens     outer fold 7/5 | tuning RF       | inner Accuracy 99.85% | o
uter Accuracy 99.90%
dataset:sens     outer fold 7/5 | tuning SVM      | inner Accuracy 99.30% | o
uter Accuracy 99.30%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
dataset:sens     outer fold 8/5 | tuning LR       | inner Accuracy 96.75% | o
uter Accuracy 96.30%
dataset:sens     outer fold 8/5 | tuning RF       | inner Accuracy 99.78% | o
uter Accuracy 100.00%
dataset:sens     outer fold 8/5 | tuning SVM      | inner Accuracy 99.22% | o
uter Accuracy 99.40%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.77 µs
dataset:sens     outer fold 9/5 | tuning LR       | inner Accuracy 96.53% | o
uter Accuracy 96.50%
dataset:sens     outer fold 9/5 | tuning RF       | inner Accuracy 99.83% | o
uter Accuracy 99.80%
dataset:sens     outer fold 9/5 | tuning SVM      | inner Accuracy 99.35% | o
uter Accuracy 99.50%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.29 µs
dataset:sens     outer fold 10/5 | tuning LR       | inner Accuracy 97.15% |
outer Accuracy 96.50%
dataset:sens     outer fold 10/5 | tuning RF       | inner Accuracy 99.78% |
outer Accuracy 99.90%
dataset:sens     outer fold 10/5 | tuning SVM      | inner Accuracy 99.30% |
outer Accuracy 99.10%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.29 µs
dataset:sens     outer fold 11/5 | tuning LR       | inner Accuracy 96.80% |
outer Accuracy 96.90%
dataset:sens     outer fold 11/5 | tuning RF       | inner Accuracy 99.83% |
outer Accuracy 99.70%
dataset:sens     outer fold 11/5 | tuning SVM      | inner Accuracy 99.25% |
outer Accuracy 98.60%
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.77 µs
dataset:sens     outer fold 12/5 | tuning LR       | inner Accuracy 96.92% |
outer Accuracy 97.50%
dataset:sens     outer fold 12/5 | tuning RF       | inner Accuracy 99.80% |
outer Accuracy 100.00%
dataset:sens     outer fold 12/5 | tuning SVM      | inner Accuracy 99.38% |
outer Accuracy 99.60%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 4.77 µs
dataset:sens     outer fold 13/5 | tuning LR       | inner Accuracy 97.08% |
outer Accuracy 97.00%
dataset:sens     outer fold 13/5 | tuning RF       | inner Accuracy 99.78% |
outer Accuracy 99.70%
dataset:sens     outer fold 13/5 | tuning SVM      | inner Accuracy 99.20% |
outer Accuracy 99.40%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 5.01 µs
dataset:sens     outer fold 14/5 | tuning LR       | inner Accuracy 96.88% |
outer Accuracy 96.60%
dataset:sens     outer fold 14/5 | tuning RF       | inner Accuracy 99.83% |
outer Accuracy 99.90%
dataset:sens     outer fold 14/5 | tuning SVM      | inner Accuracy 99.02% |
outer Accuracy 99.70%
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 4.05 µs
```

```
dataset:sens     outer fold 15/5 | tuning LR       | inner Accuracy 96.55% |
outer Accuracy 97.70%
dataset:sens     outer fold 15/5 | tuning RF       | inner Accuracy 99.78% |
outer Accuracy 99.80%
dataset:sens     outer fold 15/5 | tuning SVM      | inner Accuracy 99.45% |
outer Accuracy 99.80%
CPU times: user 5h 41min 54s, sys: 23.4 s, total: 5h 42min 17s
Wall time: 5h 16min 13s
```

In [25]:

```
for name in cv_scores:
    print('%-8s | outer CV acc. %.2f%% +\- %.3f' % (
          name, 100 * np.mean(cv_scores[name]), 100 * np.std(cv_scores[name
])))
print()
for name in cv_scores:
    print('{} best parameters'.format(name), griddles[name].best_params_)
print()
```

```
RF       | outer CV acc. 95.06% +\- 3.379
SVM      | outer CV acc. 96.64% +\- 2.335
LR       | outer CV acc. 89.99% +\- 9.266

RF best parameters {'classifier': RandomForestClassifier(bootstrap=True, clas
s_weight=None, criterion='gini',
            max_depth=None, max_features=2, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False), 'classifier__n_estimators': 1000, 'classifier_
_max_features': 2}
SVM best parameters {'classifier__C': 10.0, 'classifier__kernel': 'rbf', 'cla
ssifier__gamma': 0.01}
LR best parameters {'classifier__penalty': 'l2', 'classifier__C': 10.0}
```

In [100]:

```
for i in (['RF', 'SVM', 'LR']):
    for a,b,d,j,k in ([(letters_train_x, letters_train_y, letters_test_x, lett
ers_test_y, 'letters'),(isolet_train_x, isolet_train_y, isolet_test_x, isolet_
test_y, 'isolet'), (sens_train_x, sens_train_y, sens_test_x, sens_test_y, 'sen
s')]):
#         for t in ([''])
        best_algo = griddles[i]
        print('______________________________________________________________'
)
#best_algo = griddles['RF']
        print(i)
        best_algo.fit(a, b)
        y_predictiontr = best_algo.predict(a)
        y_predictiontest = best_algo.predict(d)
        train_acc = accuracy_score(y_true=b, y_pred=y_predictiontr)
        test_acc = accuracy_score(y_true=j, y_pred=y_predictiontest)
        train_f =f1_score(y_true=b, y_pred = y_predictiontr, average='micro')
        test_f = f1_score(y_true=j, y_pred=y_predictiontest, average='micro')
        #tf,tp,thresh = metrics.roc_curve(b,y_predictiontr, pos_label=2)
        print( 'the dataset: %s \n'% k)
        print('Inner Accuracy %.2f%% (average over CV test folds)' %
        (100 * best_algo.best_score_))
        print('Best Parameters: %s' % griddles['SVM'].best_params_)
        print('Training Accuracy: %.2f%%' % (100 * train_acc))
        print('Test Accuracy: %.2f%% \n' % (100 * test_acc))
        print('F1 training score: %.2f%% ' % (train_f))
        print('F1 testing score: %.2f%% ' % (test_f))
        print('Average split0 test score %.2f%% with std of %.5f%%' % ((np.mea
n(best_algo.cv_results_['split0_test_score'])), np.mean(best_algo.cv_results_[
'std_test_score'])))
        print('Average split0 train score %.2f%% with std of %.5f%%' % ((np.me
an(best_algo.cv_results_['split0_train_score'])), np.mean(best_algo.cv_results
_['std_train_score'])))

#         pashm = pd.DataFrame(best_algo.cv_results_)
#         print(pashm['split0_test_score','split1_test_score','split0_train_sc
ore','split1_train_score','std_test_score','std_train_score'])
```

RF

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-100-9ba14d3136c0> in <module>()
      6 #best_algo = griddles['RF']
      7         print(i)
----> 8         best_algo.fit(a, b)
      9         y_predictiontr = best_algo.predict(a)
     10         y_predictiontest = best_algo.predict(d)

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/model_selection/
_search.py in fit(self, X, y, groups, **fit_params)
    720                 return results_container[0]
    721 
--> 722             self._run_search(evaluate_candidates)
    723 
    724         results = results_container[0]

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/model_selection/
_search.py in _run_search(self, evaluate_candidates)
   1189     def _run_search(self, evaluate_candidates):
   1190         """Search all candidates in param_grid"""
-> 1191         evaluate_candidates(ParameterGrid(self.param_grid))
   1192 
   1193 

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/model_selection/
_search.py in evaluate_candidates(candidate_params)
    709                                for parameters, (train, test)
    710                                in product(candidate_params,
--> 711                                           cv.split(X, y, groups)))
    712 
    713                 all_candidate_params.extend(candidate_params)

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/parallel.py in __call__(self, iterable)
    984                 self._iterating = self._original_iterator is not None
    985 
--> 986             while self.dispatch_one_batch(iterator):
    987                 pass
    988 

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/parallel.py in dispatch_one_batch(self, iterator)
    823                 return False
    824             else:
--> 825                 self._dispatch(tasks)
    826                 return True
    827 

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/parallel.py in _dispatch(self, batch)
    780         with self._lock:
    781             job_idx = len(self._jobs)
--> 782             job = self._backend.apply_async(batch, callback=cb)
    783             # A job can complete so quickly than its callback is
    784             # called before we get here, causing self._jobs to
```

```
/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/_parallel_backends.py in apply_async(self, func, callback)
    180     def apply_async(self, func, callback=None):
    181         """Schedule a func to be run"""
--> 182         result = ImmediateResult(func)
    183         if callback:
    184             callback(result)

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/_parallel_backends.py in __init__(self, batch)
    543         # Don't delay the application, to avoid keeping the input
    544         # arguments in memory
--> 545         self.results = batch()
    546 
    547     def get(self):

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/parallel.py in __call__(self)
    259         with parallel_backend(self._backend):
    260             return [func(*args, **kwargs)
--> 261                     for func, args, kwargs in self.items]
    262 
    263     def __len__(self):

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/parallel.py in <listcomp>(.0)
    259         with parallel_backend(self._backend):
    260             return [func(*args, **kwargs)
--> 261                     for func, args, kwargs in self.items]
    262 
    263     def __len__(self):

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/model_selection/
_validation.py in _fit_and_score(estimator, X, y, scorer, train, test, verbos
e, parameters, fit_params, return_train_score, return_parameters, return_n_te
st_samples, return_times, return_estimator, error_score)
    566         fit_time = time.time() - start_time
    567         # _score will return dict if is_multimetric is True
--> 568         test_scores = _score(estimator, X_test, y_test, scorer, is_mu
ltimetric)
    569         score_time = time.time() - start_time - fit_time
    570         if return_train_score:

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/model_selection/
_validation.py in _score(estimator, X_test, y_test, scorer, is_multimetric)
    603     """
    604     if is_multimetric:
--> 605         return _multimetric_score(estimator, X_test, y_test, scorer)
    606     else:
    607         if y_test is None:

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/model_selection/
_validation.py in _multimetric_score(estimator, X_test, y_test, scorers)
    633             score = scorer(estimator, X_test)
    634         else:
--> 635             score = scorer(estimator, X_test, y_test)
    636 
```

```
    637         if hasattr(score, 'item'):

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/metrics/scorer.p
y in __call__(self, estimator, X, y_true, sample_weight)
     89         """
     90
---> 91         y_pred = estimator.predict(X)
     92         if sample_weight is not None:
     93             return self._sign * self._score_func(y_true, y_pred,

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/utils/metaestima
tors.py in <lambda>(*args, **kwargs)
    116
    117         # lambda, but not partial, allows help() to work with update_
wrapper
--> 118         out = lambda *args, **kwargs: self.fn(obj, *args, **kwargs)
    119         # update the docstring of the returned function
    120         update_wrapper(out, self.fn)

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/pipeline.py in p
redict(self, X, **predict_params)
    330             if transform is not None:
    331                 Xt = transform.transform(Xt)
--> 332         return self.steps[-1][-1].predict(Xt, **predict_params)
    333
    334     @if_delegate_has_method(delegate='_final_estimator')

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/ensemble/forest.
py in predict(self, X)
    543             The predicted classes.
    544         """
--> 545         proba = self.predict_proba(X)
    546
    547         if self.n_outputs_ == 1:

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/ensemble/forest.
py in predict_proba(self, X)
    595             delayed(_accumulate_prediction)(e.predict_proba, X, all_p
roba,
    596                                             lock)
--> 597             for e in self.estimators_)
    598
    599         for proba in all_proba:

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/parallel.py in __call__(self, iterable)
    984                 self._iterating = self._original_iterator is not None
    985
--> 986             while self.dispatch_one_batch(iterator):
    987                 pass
    988

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/parallel.py in dispatch_one_batch(self, iterator)
    823                 return False
    824             else:
--> 825                 self._dispatch(tasks)
```

```
    826                     return True
    827 

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/parallel.py in _dispatch(self, batch)
    780         with self._lock:
    781             job_idx = len(self._jobs)
--> 782             job = self._backend.apply_async(batch, callback=cb)
    783             # A job can complete so quickly than its callback is
    784             # called before we get here, causing self._jobs to

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/_parallel_backends.py in apply_async(self, func, callback)
    180     def apply_async(self, func, callback=None):
    181         """Schedule a func to be run"""
--> 182         result = ImmediateResult(func)
    183         if callback:
    184             callback(result)

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/_parallel_backends.py in __init__(self, batch)
    543         # Don't delay the application, to avoid keeping the input
    544         # arguments in memory
--> 545         self.results = batch()
    546 
    547     def get(self):

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/parallel.py in __call__(self)
    259         with parallel_backend(self._backend):
    260             return [func(*args, **kwargs)
--> 261                     for func, args, kwargs in self.items]
    262 
    263     def __len__(self):

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/externals/jobli
b/parallel.py in <listcomp>(.0)
    259         with parallel_backend(self._backend):
    260             return [func(*args, **kwargs)
--> 261                     for func, args, kwargs in self.items]
    262 
    263     def __len__(self):

/home/arashidi/anaconda3/lib/python3.5/site-packages/sklearn/ensemble/forest.
py in _accumulate_prediction(predict, X, out, lock)
    392     with lock:
    393         if len(out) == 1:
--> 394             out[0] += prediction
    395         else:
    396             for i in range(len(out)):

KeyboardInterrupt: 
```

In [30]:
```
%matplotlib inline
for i in (['SVM', 'LR']):
    for a,b,d,j,k in ([(letters_train_x, letters_train_y, letters_test_x, lett
ers_test_y, 'letters'),(isolet_train_x, isolet_train_y, isolet_test_x, isolet_
test_y, 'isolet'), (sens_train_x, sens_train_y, sens_test_x, sens_test_y, 'sen
s')]):
        print("testing on %s " % k)
        best_algo = griddles[i]
        if k=='letters':
            best_algo = griddles[i]
            print('___________________________________________________________
___')
#best_algo = griddles['RF']
            print(i)
            best_algo.fit(a, b)
            y_predictiontr = best_algo.predict(a)
            y_predictiontest = best_algo.predict(d)
            train_acc = accuracy_score(y_true=b, y_pred=y_predictiontr)
            test_acc = accuracy_score(y_true=j, y_pred=y_predictiontest)
            train_f =f1_score(y_true=b, y_pred = y_predictiontr, average='micr
o')
            test_f = f1_score(y_true=j, y_pred=y_predictiontest, average='micr
o')
        #tf,tp,thresh = metrics.roc_curve(b,y_predictiontr, pos_label=2)
            print( 'the dataset: %s \n'% k)
            print('Inner Accuracy %.2f%% (average over CV test folds)' %
            (100 * best_algo.best_score_))
            print('Best Parameters: %s' % griddles['SVM'].best_params_)
            print('Training Accuracy: %.2f%%' % (100 * train_acc))
            print('Test Accuracy: %.2f%% \n' % (100 * test_acc))
            print('F1 training score: %.2f%% ' % (train_f))
            print('F1 testing score: %.2f%% ' % (test_f))
            print('Average split0 test score %.2f%% with std of %.5f%%' % ((np
.mean(best_algo.cv_results_['split0_test_score'])), np.mean(best_algo.cv_resul
ts_['std_test_score'])))
            print('Average split0 train score %.2f%% with std of %.5f%%' % ((n
p.mean(best_algo.cv_results_['split0_train_score'])), np.mean(best_algo.cv_res
ults_['std_train_score'])))
            continue
        print('______________________________________________________________'
)
        print(i)

        best_algo.fit(a, b)



        y_predictiontr = best_algo.predict(a)
        y_predictiontest = best_algo.predict(d)


        train_acc = accuracy_score(y_true=b, y_pred=y_predictiontr)
        test_acc = accuracy_score(y_true=j, y_pred=y_predictiontest)
        print('after training accuracy')

        train_f =f1_score(y_true=b, y_pred = y_predictiontr, average='micro')
```

```
        test_f = f1_score(y_true=j, y_pred=y_predictiontest, average='micro')
        tr_sizes=[1,500,1000,2000,2500,3000,3500,4000]
        train_sizes, train_scores, test_scores= learning_curve(LinearRegressio
n(),a,b,train_sizes=tr_sizes, cv=5, scoring='neg_mean_squared_error')
        train_scores_mean = -train_scores.mean(axis=1)
        test_scores_mean = -test_scores.mean(axis=1)
        print('after scores mean')

#         print(train_scores_mean)
#         print(test_scores_mean)

        # best_algo.fit(a, b)
#         plt.style.use('seaborn')

        plt.plot(train_sizes, train_scores_mean, label='training error % for '
+k +' in ' + i)
        plt.legend()
        plt.savefig(k+i+'.png')
        plt.clf()
        # plt.show()
        plt.plot(train_sizes, test_scores_mean, label='testing error % for '+
k + ' in '+i)
        plt.legend()
        plt.savefig(k+i+'.png')
        plt.clf()
        mse_scorer = make_scorer(mean_squared_error)


        # mse_tr = mean_squared_error(y_true=b, y_pred = y_predictiontr)
        # mse_test= mean_squared_error(j, y_predictiontest)
        #tf,tp,thresh = metrics.roc_curve(b,y_predictiontr, pos_label=2)


        print('The dataset: %s \n' % k)
        print('Average split0 test score %.2f%% with std of %.5f%%' % ((np.mea
n(best_algo.cv_results_['split0_test_score'])), np.mean(best_algo.cv_results_[
'std_test_score'])))
        print('Average split0 train score %.2f%% with std of %.5f%%' % ((np.me
an(best_algo.cv_results_['split0_train_score'])), np.mean(best_algo.cv_results
_['std_train_score'])))
        print('Inner Accuracy %.2f%% (average over 5 CV test folds)' %
        (100 * best_algo.best_score_))
#         print('Cross validation score')
#         print( cross_val_score(best_algo, a,b,cv=5,scoring='neg_mean_squared
_error').mean())
        print('(average over 5 CV training folds)')
        print('Cross validation score ')
#         print(cross_val_score(best_algo, d,j,cv=5,scoring='neg_mean_squared_
error').mean())
        print('(average over 5 CV testing folds)')
        print('Best Parameters: %s' % griddles[i].best_params_)
        print('Training Accuracy: %.2f%%' % (100 * train_acc))
        print('Test Accuracy: %.2f%% \n' % (100 * test_acc))
        print('F1 training score: %.2f%% ' % train_f)
        print('F1 testing score: %.2f%% ' % test_f)
#         print('average precision training score: %.2f%% ' % aps_tr)
```

```
#           print('average precision testing score: %.2f%% ' % aps_test)
#           print(best_algo.cv_results_)
```

```
testing on letters
```

```
SVM
the dataset: letters

Inner Accuracy 91.30% (average over CV test folds)
Best Parameters: {'classifier__C': 10.0, 'classifier__kernel': 'rbf', 'classi
fier__gamma': 0.1}
Training Accuracy: 99.92%
Test Accuracy: 94.47%

F1 training score: 1.00%
F1 testing score: 0.94%
Average split0 test score 0.40% with std of 0.00337%
Average split0 train score 0.43% with std of 0.00252%
testing on isolet
```

```
SVM
after training accuracy
after scores mean
The dataset: isolet

Average split0 test score 0.72% with std of 0.07239%
Average split0 train score 0.93% with std of 0.10874%
Inner Accuracy 95.90% (average over 5 CV test folds)
(average over 5 CV training folds)
Cross validation score
(average over 5 CV testing folds)
Best Parameters: {'classifier__C': 10.0, 'classifier__kernel': 'rbf', 'classi
fier__gamma': 0.001}
Training Accuracy: 100.00%
Test Accuracy: 96.53%

F1 training score: 1.00%
F1 testing score: 0.97%
testing on sens
```

```
SVM
after training accuracy
after scores mean
The dataset: sens

Average split0 test score 0.83% with std of 0.01726%
Average split0 train score 0.85% with std of 0.01477%
Inner Accuracy 99.40% (average over 5 CV test folds)
(average over 5 CV training folds)
Cross validation score
(average over 5 CV testing folds)
Best Parameters: {'classifier__C': 10.0, 'classifier__kernel': 'rbf', 'classi
fier__gamma': 0.01}
Training Accuracy: 99.90%
Test Accuracy: 99.33%

F1 training score: 1.00%
F1 testing score: 0.99%
testing on letters
```

```
LR
the dataset: letters

Inner Accuracy 75.44% (average over CV test folds)
Best Parameters: {'classifier__C': 10.0, 'classifier__kernel': 'rbf', 'classi
fier__gamma': 0.01}
Training Accuracy: 78.66%
Test Accuracy: 76.80%

F1 training score: 0.79%
F1 testing score: 0.77%
Average split0 test score 0.64% with std of 0.00749%
Average split0 train score 0.67% with std of 0.00221%
testing on isolet
```

```
LR
after training accuracy
after scores mean
The dataset: isolet

Average split0 test score 0.93% with std of 0.00189%
Average split0 train score 0.98% with std of 0.00099%
Inner Accuracy 95.14% (average over 5 CV test folds)
(average over 5 CV training folds)
Cross validation score
(average over 5 CV testing folds)
Best Parameters: {'classifier__penalty': 'l2', 'classifier__C': 1.0}
Training Accuracy: 100.00%
Test Accuracy: 94.91%

F1 training score: 1.00%
F1 testing score: 0.95%
testing on sens
```

```
LR
after training accuracy
after scores mean
The dataset: sens

Average split0 test score 0.90% with std of 0.00387%
Average split0 train score 0.92% with std of 0.00657%
Inner Accuracy 96.92% (average over 5 CV test folds)
(average over 5 CV training folds)
Cross validation score
(average over 5 CV testing folds)
Best Parameters: {'classifier__penalty': 'l2', 'classifier__C': 10.0}
Training Accuracy: 97.82%
Test Accuracy: 97.41%

F1 training score: 0.98%
F1 testing score: 0.97%
```

```
<matplotlib.figure.Figure at 0x7f414f62f780>
```

In [94]:
```
sklearn.metrics.SCORERS.keys()
```

Out[94]:
```
dict_keys(['mutual_info_score', 'neg_median_absolute_error', 'brier_score_los
s', 'normalized_mutual_info_score', 'precision', 'recall_micro', 'homogeneity
_score', 'completeness_score', 'f1', 'recall', 'average_precision', 'fowlkes_
mallows_score', 'f1_macro', 'neg_mean_squared_log_error', 'precision_micro',
'r2', 'neg_log_loss', 'recall_samples', 'v_measure_score', 'f1_micro', 'adjus
ted_mutual_info_score', 'recall_macro', 'recall_weighted', 'balanced_accurac
y', 'neg_mean_absolute_error', 'precision_macro', 'f1_weighted', 'explained_v
ariance', 'neg_mean_squared_error', 'precision_weighted', 'roc_auc', 'precisi
on_samples', 'adjusted_rand_score', 'f1_samples', 'accuracy'])
```

In [81]:
```
# test=pd.DataFrame(columns=['mean_fit_time', 'mean_score_time', 'mean_test_sc
ore',
#        'mean_train_score', 'param_classifier__C', 'param_classifier__penalt
y',
#        'params', 'rank_test_score', 'split0_test_score', 'split0_train_scor
e',
#        'split1_test_score', 'split1_train_score', 'std_fit_time',
#        'std_score_time', 'std_test_score', 'std_train_score' ])
for i in (['LR']):
    for a,b,d,j,k in ([(letters_train_x, letters_train_y, letters_test_x, lett
ers_test_y, 'letters'),(isolet_train_x, isolet_train_y, isolet_test_x, isolet_
test_y, 'isolet'), (sens_train_x, sens_train_y, sens_test_x, sens_test_y, 'sen
s')]):
#         for t in ([''])
        best_algo = griddles[i]
        print('______________________________________________________________'
)
#best_algo = griddles['RF']
        print(i)
        best_algo.fit(a, b)
#         y_predictiontr = best_algo.predict(a)
#         y_predictiontest = best_algo.predict(d)

#         pashm = pd.DataFrame(best_algo.cv_results_, columns=['mean_fit_tim
e', 'mean_score_time', 'mean_test_score',
#        'mean_train_score', 'param_classifier__C', 'param_classifier__penalt
y',
#        'params', 'rank_test_score', 'split0_test_score', 'split0_train_scor
e',
#        'split1_test_score', 'split1_train_score', 'std_fit_time',
#        'std_score_time', 'std_test_score', 'std_train_score' ])
#        # print(pashm)
#         a = [pd.DataFrame(best_algo.cv_results_['params']),pd.DataFrame(best
_algo.cv_results_['std_test_score'])]
#         result = pd.concat([test, pd.DataFrame(a)])
#         print(result)
#         test.append(pd.DataFrame(best_algo.cv_results_))
        test=pd.DataFrame(best_algo.cv_results_)
        print(test)
#         print(best_algo.cv_results_['params'])
        print('_____after________')
#         print(test)

#         print(pashm.columns)
#         test.append(pashm,ignore_index=True)
#         print(test)
        #         print(pashm['split0_test_score','split1_test_score','split0_
train_score','split1_train_score','std_test_score','std_train_score'])
```

```
LR
   mean_fit_time  mean_score_time  mean_test_score  mean_train_score  \
0       0.059131         0.003785           0.2572          0.265008
1       0.071548         0.003860           0.4940          0.508412
2       0.120892         0.003896           0.6360          0.659193
3       0.180794         0.003888           0.7252          0.757006
4       0.249718         0.003894           0.7516          0.792815
5       0.304410         0.003880           0.7544          0.796200
6       0.354364         0.003950           0.7538          0.795804
7       0.373304         0.003929           0.7534          0.796205

  param_classifier__C param_classifier__penalty  \
0              0.0001                        l2
1               0.001                        l2
2                0.01                        l2
3                 0.1                        l2
4                   1                        l2
5                  10                        l2
6                 100                        l2
7                1000                        l2

                                              params  rank_test_score  \
0  {'classifier__C': 0.0001, 'classifier__penalty...                8
1  {'classifier__C': 0.001, 'classifier__penalty'...                7
2  {'classifier__C': 0.01, 'classifier__penalty':...                6
3  {'classifier__C': 0.1, 'classifier__penalty': ...                5
4  {'classifier__C': 1.0, 'classifier__penalty': ...                4
5  {'classifier__C': 10.0, 'classifier__penalty':...                1
6  {'classifier__C': 100.0, 'classifier__penalty'...                2
7  {'classifier__C': 1000.0, 'classifier__penalty...                3

   split0_test_score  split0_train_score  split1_test_score  \
0           0.264354            0.267657           0.250000
1           0.487640            0.512039           0.500401
2           0.630781            0.656902           0.641252
3           0.717703            0.758828           0.732745
4           0.744817            0.797352           0.758427
5           0.744418            0.796148           0.764446
6           0.745215            0.796950           0.762440
7           0.745215            0.797753           0.761637

   split1_train_score  std_fit_time  std_score_time  std_test_score  \
0            0.262360      0.003015    1.549721e-05        0.007177
1            0.504785      0.000437    4.005432e-05        0.006381
2            0.661483      0.002331    3.576279e-07        0.005235
3            0.755183      0.001055    4.649162e-06        0.007521
4            0.788278      0.008404    2.622604e-06        0.006805
5            0.796252      0.002902    1.990795e-05        0.010014
6            0.794657      0.011964    2.360344e-05        0.008612
7            0.794657      0.017015    3.695488e-06        0.008211

   std_train_score
0         0.002648
1         0.003627
2         0.002291
3         0.001822
```

```
4         0.004537
5         0.000052
6         0.001147
7         0.001548
_____after________
______________________________________________________________
LR
   mean_fit_time  mean_score_time  mean_test_score  mean_train_score  \
0       0.438834         0.009486           0.8550          0.882586
1       0.938920         0.009778           0.9206          0.951999
2       1.366928         0.009876           0.9474          0.992599
3       1.869666         0.009908           0.9512          1.000000
4       2.255748         0.009770           0.9514          1.000000
5       2.168054         0.009948           0.9498          1.000000
6       1.864004         0.010162           0.9500          1.000000
7       1.847563         0.009950           0.9492          1.000000

  param_classifier__C param_classifier__penalty  \
0              0.0001                        l2
1               0.001                        l2
2                0.01                        l2
3                 0.1                        l2
4                   1                        l2
5                  10                        l2
6                 100                        l2
7                1000                        l2

                                              params  rank_test_score  \
0  {'classifier__C': 0.0001, 'classifier__penalty...                8
1  {'classifier__C': 0.001, 'classifier__penalty'...                7
2  {'classifier__C': 0.01, 'classifier__penalty':...                6
3  {'classifier__C': 0.1, 'classifier__penalty': ...                2
4  {'classifier__C': 1.0, 'classifier__penalty': ...                1
5  {'classifier__C': 10.0, 'classifier__penalty':...                4
6  {'classifier__C': 100.0, 'classifier__penalty'...                3
7  {'classifier__C': 1000.0, 'classifier__penalty...                5

   split0_test_score  split0_train_score  split1_test_score  \
0           0.843513            0.875752           0.866533
1           0.919760            0.951503           0.921443
2           0.947305            0.991984           0.947495
3           0.952096            1.000000           0.950301
4           0.952096            1.000000           0.950701
5           0.950100            1.000000           0.949499
6           0.950100            1.000000           0.949900
7           0.948503            1.000000           0.949900

   split1_train_score  std_fit_time  std_score_time  std_test_score  \
0            0.889421      0.011492        0.000304        0.011510
1            0.952495      0.033112        0.000214        0.000841
2            0.993214      0.047729        0.000096        0.000095
3            1.000000      0.016810        0.000055        0.000898
4            1.000000      0.243287        0.000096        0.000697
5            1.000000      0.103235        0.000158        0.000300
6            1.000000      0.120761        0.000134        0.000100
7            1.000000      0.252671        0.000055        0.000698
```

```
   std_train_score
0         0.006835
1         0.000496
2         0.000615
3         0.000000
4         0.000000
5         0.000000
6         0.000000
7         0.000000
_____after________
______________________________________________________________
LR
   mean_fit_time  mean_score_time  mean_test_score  mean_train_score  \
0       0.045855         0.001775           0.6878          0.697617
1       0.060075         0.001963           0.7934          0.803212
2       0.082609         0.001992           0.8820          0.896815
3       0.113160         0.001977           0.9432          0.956604
4       0.153323         0.001971           0.9646          0.974002
5       0.249031         0.001962           0.9692          0.982403
6       0.549309         0.001975           0.9680          0.985805
7       0.971581         0.001950           0.9668          0.987405

  param_classifier__C param_classifier__penalty  \
0              0.0001                        l2
1               0.001                        l2
2                0.01                        l2
3                 0.1                        l2
4                   1                        l2
5                  10                        l2
6                 100                        l2
7                1000                        l2

                                              params  rank_test_score  \
0  {'classifier__C': 0.0001, 'classifier__penalty...                8
1  {'classifier__C': 0.001, 'classifier__penalty'...                7
2  {'classifier__C': 0.01, 'classifier__penalty':...                6
3  {'classifier__C': 0.1, 'classifier__penalty': ...                5
4  {'classifier__C': 1.0, 'classifier__penalty': ...                4
5  {'classifier__C': 10.0, 'classifier__penalty':...                1
6  {'classifier__C': 100.0, 'classifier__penalty'...                2
7  {'classifier__C': 1000.0, 'classifier__penalty...                3

   split0_test_score  split0_train_score  split1_test_score  \
0           0.697163            0.712054           0.678414
1           0.795046            0.813376           0.791750
2           0.893328            0.909091           0.870645
3           0.946864            0.960352           0.939527
4           0.962845            0.975571           0.966360
5           0.968038            0.984782           0.970364
6           0.966440            0.989588           0.969563
7           0.967239            0.991590           0.966360

   split1_train_score  std_fit_time  std_score_time  std_test_score  \
0            0.683180      0.006176        0.000034        0.009375
1            0.793048      0.000836        0.000024        0.001648
2            0.884539      0.004613        0.000024        0.011342
3            0.952857      0.022260        0.000028        0.003668
```

```
4            0.972433     0.032068      0.000018      0.001758
5            0.980024     0.058410      0.000007      0.001163
6            0.982022     0.092038      0.000012      0.001562
7            0.983220     0.218380      0.000008      0.000440

   std_train_score
0         0.014437
1         0.010164
2         0.012276
3         0.003748
4         0.001569
5         0.002379
6         0.003783
7         0.004185
_____after________
```