Anteriormente habíamos comprovado cómo es interesante usar PCA después de la extracción de features aleatorios. Ahora vamos a ver cómo se comporta con RandomForest, pero vamos a usar un dataset más adecuado que antes, uno que solo tiene variables numéricas.

En principio esto solo hay que implementarlo en python

In [1]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

In [2]:
```
path = "../../datasets/falldetection/falldetection.csv"
```

In [3]:
```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.kernel_approximation import RBFSampler
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import math
import ipywidgets as widgets
from IPython.display import clear_output
#from markdown import markdown as md
from IPython.display import Markdown as md
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import scale
from sklearn.kernel_approximation import Nystroem
#df = pd.read_csv(path)
```

In [4]:
```
def get_data(d, prop_train = 2/3, perform_normalization = True):
    dsize = size_selector.value
    if d == "fall_detection":
        df = pd.read_csv(path)
        #N = df.shape[0]
        if dsize == -1: dsize = df.shape[0]
        N = min(df.shape[0], dsize)
        df = df.sample(n = N)

        #prop_train = 2 / 3
        N_train = np.ceil(N * prop_train).astype(np.int64)
        N_test = N - N_train

        data = df.drop(["ACTIVITY"], 1)
        target = df.ACTIVITY


        if perform_normalization:
            data = pd.DataFrame(scale(data.astype(np.float64)))

        data_train = data.iloc[:N_train]
        data_test = data.iloc[N_train:]

        target_train = target[:N_train]
        target_test = target[N_train:]

        return data_train, data_test, target_train, target_test
    elif d == "digits":
        digits = datasets.load_digits()
        target = digits.target
        data = digits.data

        #data /= 16
        #data -= data.mean(axis=0)
        if perform_normalization:
            data = scale(data.astype(np.float64))

        N = data.shape[0]
        #prop_train = 2 / 3
        N_train = math.ceil(N * prop_train)
        N_test = N - N_train

        data_train = data[:N_train]
        data_test = data[N_train:]

        target_train = target[:N_train]
        target_test = target[N_train:]

        return data_train, data_test, target_train, target_test
    elif d == "full_covertype":
        p = "../../datasets/covertype/full_covertype_cleaned.csv"
        df = pd.read_csv(p)

        if dsize == -1: dsize = df.shape[0]
        N = min(df.shape[0], dsize)
        df = df.sample(n = N)

        #N = data.shape[0]
        #prop_train = 2 / 3
        N_train = np.ceil(N * prop_train).astype(np.int64)
        N_test = N - N_train

        target = df.target
        data = df.drop(labels = "target", axis = 1)

        if perform_normalization:
            data = pd.DataFrame(scale(data.astype(np.float64)))
```

N = df.shape[0]

prop_train = 2 / 3 N_train = np.ceil(N * prop_train).astype(np.int64) N_test = N - N_train

data = df.drop(["ACTIVITY"], 1) target = df.ACTIVITY

data_train = data.iloc[:N_train] data_test = data.iloc[N_train:]

target_train = target[:N_train] target_test = target[N_train:]

In [5]:
```
def linearSVM_graph():
    SVM_clf = LinearSVC(max_iter=999999)
    data_train, data_test, target_train, target_test = get_data(dataset_selec

    sub_progress_bar.min = 0
    sub_progress_bar.max = 2
    sub_progress_bar.value = 0

    SVM_clf.fit(data_train, target_train)

    sub_progress_bar.value += 1

    SVM_train_score = SVM_clf.score(data_train, target_train)
    SVM_test_score = SVM_clf.score(data_test, target_test)

    sub_progress_bar.value += 1

    train_dic = {
        'absi': [feats[0], feats[-1]],
        'ord': [SVM_train_score, SVM_train_score],
        'label': 'LinearSVM Train score'
    }

    test_dic = {
        'absi': [feats[0], feats[-1]],
        'ord': [SVM_test_score, SVM_test_score],
        'label': 'SVM Test score'
    }
    return train_dic, test_dic
    #return DT train score, DT test score
```

In [6]:
```
def logit_graph():
    logit_clf = LogisticRegression(C = 1, multi_class = 'multinomial', solver
    data_train, data_test, target_train, target_test = get_data(dataset_selec

    sub_progress_bar.min = 0
    sub_progress_bar.max = 2
    sub_progress_bar.value = 0

    logit_clf.fit(data_train, target_train)

    sub_progress_bar.value += 1

    logit_train_score = logit_clf.score(data_train, target_train)
    logit_test_score = logit_clf.score(data_test, target_test)

    sub_progress_bar.value += 1

    train_dic = {
        'absi': [feats[0], feats[-1]],
        'ord': [logit_train_score, logit_train_score],
        'label': 'Logit Train score'
    }

    test_dic = {
        'absi': [feats[0], feats[-1]],
        'ord': [logit_test_score, logit_test_score],
        'label': 'Logit Test score'
    }
    return train_dic, test_dic
```

In [7]:
```
def dt_graph():
    DT_clf = DecisionTreeClassifier()
    data_train, data_test, target_train, target_test = get_data(dataset_selec
    DT_clf.fit(data_train, target_train)
    DT_train_score = DT_clf.score(data_train, target_train)
    DT_test_score = DT_clf.score(data_test, target_test)

    train_dic = {
        'absi': [feats[0], feats[-1]],
        'ord': [DT_train_score, DT_train_score],
        'label': 'DT Train score'
    }

    test_dic = {
        'absi': [feats[0], feats[-1]],
        'ord': [DT_test_score, DT_test_score],
        'label': 'DT Test score'
    }
    return train_dic, test_dic
    #return DT_train_score, DT_test_score
```

In [8]:
```
def rf_graph():
    RF_clf = RandomForestClassifier(n_estimators = n_estimators_selector.valu
    data_train, data_test, target_train, target_test = get_data(dataset_selec

    sub_progress_bar.min = 0
    sub_progress_bar.max = 2
    sub_progress_bar.value = 0

    RF_clf.fit(data_train, target_train)

    sub_progress_bar.value += 1

    RF_train_score = RF_clf.score(data_train, target_train)
    RF_test_score = RF_clf.score(data_test, target_test)

    sub_progress_bar.value += 1

    train_dic = {
        'absi': [feats[0], feats[-1]],
        'ord': [RF_train_score, RF_train_score],
        'label': 'RF Train score'
    }

    test_dic = {
        'absi': [feats[0], feats[-1]],
        'ord': [RF_test_score, RF_test_score],
        'label': 'RF Test score'
    }
    return train_dic, test_dic
    #return RF train score, RF test score
```

In [9]:
```
def nystroem_pca_rf_blackbox_graph():
    NYS_PCA_RF_clf = Pipeline([
        #("sampler",RBFSampler(gamma = 0.2)),
        ("sampler",Nystroem(gamma=0.2)),
        ("pca", PCA(n_components = 0.9, svd_solver = "full")),
        ("clf", RandomForestClassifier(n_estimators = n_estimators_selector.v
    ])

    NYS_PCA_RF_train_scores = []
    NYS_PCA_RF_test_scores = []
    data_train, data_test, target_train, target_test = get_data(dataset_selec

    sub_progress_bar.min = 0
    sub_progress_bar.max = len(feats)
    sub_progress_bar.value = 0

    for f in feats:
        NYS_PCA_RF_clf.set_params(sampler__n_components = f)
        NYS_PCA_RF_clf.fit(data_train, target_train)
        train_score = NYS_PCA_RF_clf.score(data_train, target_train)
        test_score = NYS_PCA_RF_clf.score(data_test, target_test)

        NYS_PCA_RF_train_scores.append(train_score)
        NYS_PCA_RF_test_scores.append(test_score)

        sub_progress_bar.value += 1


    train_dic = {
        'absi': feats,
        'ord': NYS_PCA_RF_train_scores,
        'label': 'NYS_PCA_RF (Black Box)Train scores'
    }

    test_dic = {
        'absi': feats,
        'ord': NYS_PCA_RF_test_scores,
        'label': 'NYS_PCA_RF (Black Box) Test scores'
    }
    return train_dic, test_dic
    #return RBF_PCA_RF_train_scores, RBF_PCA_RF_test_scores
```

In [10]:

```
def rbf_pca_rf_blackbox_graph():
    RBF_PCA_RF_clf = Pipeline([
        ("sampler",RBFSampler(gamma = 0.2)),
        ("pca", PCA(n_components = 0.9, svd_solver = "full")),
        ("clf", RandomForestClassifier(n_estimators = n_estimators_selector.v
    ])

    RBF_PCA_RF_train_scores = []
    RBF_PCA_RF_test_scores = []
    data_train, data_test, target_train, target_test = get_data(dataset_selec

    sub_progress_bar.min = 0
    sub_progress_bar.max = len(feats)
    sub_progress_bar.value = 0

    for f in feats:
        RBF_PCA_RF_clf.set_params(sampler__n_components = f)
        RBF_PCA_RF_clf.fit(data_train, target_train)
        train_score = RBF_PCA_RF_clf.score(data_train, target_train)
        test_score = RBF_PCA_RF_clf.score(data_test, target_test)

        RBF_PCA_RF_train_scores.append(train_score)
        RBF_PCA_RF_test_scores.append(test_score)

        sub_progress_bar.value += 1


    train_dic = {
        'absi': feats,
        'ord': RBF_PCA_RF_train_scores,
        'label': 'RBF_PCA_RF (Black Box)Train scores'
    }

    test_dic = {
        'absi': feats,
        'ord': RBF_PCA_RF_test_scores,
        'label': 'RBF_PCA_RF (Black Box) Test scores'
    }
    return train_dic, test_dic
    #return RBF_PCA_RF_train_scores, RBF_PCA_RF_test_scores
```

In [11]:
```
def nystroem_pca_dt_graph():
    NYS_PCA_DT_clf = Pipeline([
        ("sampler",Nystroem(gamma = 0.2)),
        ("pca", PCA(n_components = 0.9, svd_solver = "full")),
        ("clf", DecisionTreeClassifier())
    ])

    NYS_PCA_DT_train_scores = []
    NYS_PCA_DT_test_scores = []
    data_train, data_test, target_train, target_test = get_data(dataset_selec

    sub_progress_bar.min = 0
    sub_progress_bar.max = len(feats)
    sub_progress_bar.value = 0

    for f in feats:
        NYS_PCA_DT_clf.set_params(sampler__n_components = f)
        NYS_PCA_DT_clf.fit(data_train, target_train)
        train_score = NYS_PCA_DT_clf.score(data_train, target_train)
        test_score = NYS_PCA_DT_clf.score(data_test, target_test)

        NYS_PCA_DT_train_scores.append(train_score)
        NYS_PCA_DT_test_scores.append(test_score)

        sub_progress_bar.value += 1


    train_dic = {
        'absi': feats,
        'ord': NYS_PCA_DT_train_scores,
        'label': 'NYS_PCA_DT Train scores'
    }

    test_dic = {
        'absi': feats,
        'ord': NYS_PCA_DT_test_scores,
        'label': 'NYS_PCA_DT Test scores'
    }
    return train_dic, test_dic
    #return RBF_PCA_DT_train_scores, RBF_PCA_DT_test_scores
```

In [12]:
```
def rbf_pca_dt_graph():
    RBF_PCA_DT_clf = Pipeline([
        ("sampler",RBFSampler(gamma = 0.2)),
        ("pca", PCA(n_components = 0.9, svd_solver = "full")),
        ("clf", DecisionTreeClassifier())
    ])

    RBF_PCA_DT_train_scores = []
    RBF_PCA_DT_test_scores = []
    data_train, data_test, target_train, target_test = get_data(dataset_selec

    sub_progress_bar.min = 0
    sub_progress_bar.max = len(feats)
    sub_progress_bar.value = 0

    for f in feats:
        RBF_PCA_DT_clf.set_params(sampler__n_components = f)
        RBF_PCA_DT_clf.fit(data_train, target_train)
        train_score = RBF_PCA_DT_clf.score(data_train, target_train)
        test_score = RBF_PCA_DT_clf.score(data_test, target_test)

        RBF_PCA_DT_train_scores.append(train_score)
        RBF_PCA_DT_test_scores.append(test_score)

        sub_progress_bar.value += 1


    train_dic = {
        'absi': feats,
        'ord': RBF_PCA_DT_train_scores,
        'label': 'RBF_PCA_DT Train scores'
    }

    test_dic = {
        'absi': feats,
        'ord': RBF_PCA_DT_test_scores,
        'label': 'RBF_PCA_DT Test scores'
    }
    return train_dic, test_dic
    #return RBF_PCA_DT_train_scores, RBF_PCA_DT_test_scores
```

In [13]:
```
def nystroem_pca_linearSVM_greybox_graph():
    sub_clf = Pipeline([
        ("sampler",Nystroem(gamma = 0.2)),
        ("pca", PCA(n_components = 0.9, svd_solver = "full")),
        ("clf", LinearSVC(C = 1))
    ])

    NYS_PCA_linearSVM_clf = BaggingClassifier(
        base_estimator = sub_clf,
        n_estimators = n_estimators_selector.value,
        bootstrap = True)


    NYS_PCA_linearSVM_train_scores = []
    NYS_PCA_linearSVM_test_scores = []

    sub_progress_bar.min = 0
    sub_progress_bar.max = len(feats)
    sub_progress_bar.value = 0

    data_train, data_test, target_train, target_test = get_data(dataset_selec
    for f in feats:
        NYS_PCA_linearSVM_clf.set_params(base_estimator__sampler__n_component
        NYS_PCA_linearSVM_clf.fit(data_train, target_train)
        train_score = NYS_PCA_linearSVM_clf.score(data_train, target_train)
        test_score = NYS_PCA_linearSVM_clf.score(data_test, target_test)

        NYS_PCA_linearSVM_train_scores.append(train_score)
        NYS_PCA_linearSVM_test_scores.append(test_score)

        sub_progress_bar.value += 1


    train_dic = {
        'absi': feats,
        'ord': NYS_PCA_linearSVM_train_scores,
        'label': 'NYS_PCA_LinearSVM (Grey Box)Train scores'
    }

    test_dic = {
        'absi': feats,
        'ord': NYS_PCA_linearSVM_test_scores,
        'label': 'NYS_PCA_LinearSVM (Grey Box) Test scores'
    }
    return train_dic, test_dic
```

In [14]:
```
def rbf_pca_linearSVM_greybox_graph():
    sub_clf = Pipeline([
        ("sampler",RBFSampler(gamma = 0.2)),
        ("pca", PCA(n_components = 0.9, svd_solver = "full")),
        ("clf", LinearSVC(C = 1))
    ])

    RBF_PCA_linearSVM_clf = BaggingClassifier(
        base_estimator = sub_clf,
        n_estimators = n_estimators_selector.value,
        bootstrap = True)


    RBF_PCA_linearSVM_train_scores = []
    RBF_PCA_linearSVM_test_scores = []

    sub_progress_bar.min = 0
    sub_progress_bar.max = len(feats)
    sub_progress_bar.value = 0

    data_train, data_test, target_train, target_test = get_data(dataset_selec
    for f in feats:
        RBF_PCA_linearSVM_clf.set_params(base_estimator__sampler__n_component
        RBF_PCA_linearSVM_clf.fit(data_train, target_train)
        train_score = RBF_PCA_linearSVM_clf.score(data_train, target_train)
        test_score = RBF_PCA_linearSVM_clf.score(data_test, target_test)

        RBF_PCA_linearSVM_train_scores.append(train_score)
        RBF_PCA_linearSVM_test_scores.append(test_score)

        sub_progress_bar.value += 1


    train_dic = {
        'absi': feats,
        'ord': RBF_PCA_linearSVM_train_scores,
        'label': 'RBF_PCA_LinearSVM (Grey Box)Train scores'
    }

    test_dic = {
        'absi': feats,
        'ord': RBF_PCA_linearSVM_test_scores,
        'label': 'RBF_PCA_LinearSVM (Grey Box) Test scores'
    }
    return train_dic, test_dic
```

In [15]:
```
def nystroem_pca_logit_greybox_graph():
    sub_clf = Pipeline([
        ("sampler",Nystroem(gamma = 0.2)),
        ("pca", PCA(n_components = 0.9, svd_solver = "full")),
        ("clf", LogisticRegression(C = 1, multi_class = 'multinomial', solver
    ])

    NYS_PCA_logit_clf = BaggingClassifier(
        base_estimator = sub_clf,
        n_estimators = n_estimators_selector.value,
        bootstrap = True)


    NYS_PCA_logit_train_scores = []
    NYS_PCA_logit_test_scores = []

    sub_progress_bar.min = 0
    sub_progress_bar.max = len(feats)
    sub_progress_bar.value = 0

    data_train, data_test, target_train, target_test = get_data(dataset_selec
    for f in feats:
        NYS_PCA_logit_clf.set_params(base_estimator__sampler__n_components =
        NYS_PCA_logit_clf.fit(data_train, target_train)
        train_score = NYS_PCA_logit_clf.score(data_train, target_train)
        test_score = NYS_PCA_logit_clf.score(data_test, target_test)

        NYS_PCA_logit_train_scores.append(train_score)
        NYS_PCA_logit_test_scores.append(test_score)

        sub_progress_bar.value += 1


    train_dic = {
        'absi': feats,
        'ord': NYS_PCA_logit_train_scores,
        'label': 'NYS_PCA_Logit (Grey Box)Train scores'
    }

    test_dic = {
        'absi': feats,
        'ord': NYS_PCA_logit_test_scores,
        'label': 'NYS_PCA_Logit (Grey Box) Test scores'
    }
    return train_dic, test_dic
```

In [16]:
```
def rbf_pca_logit_greybox_graph():
    sub_clf = Pipeline([
        ("sampler",RBFSampler(gamma = 0.2)),
        ("pca", PCA(n_components = 0.9, svd_solver = "full")),
        ("clf", LogisticRegression(C = 1, multi_class = 'multinomial', solver
    ])

    RBF_PCA_logit_clf = BaggingClassifier(
        base_estimator = sub_clf,
        n_estimators = n_estimators_selector.value,
        bootstrap = True)


    RBF_PCA_logit_train_scores = []
    RBF_PCA_logit_test_scores = []

    sub_progress_bar.min = 0
    sub_progress_bar.max = len(feats)
    sub_progress_bar.value = 0

    data_train, data_test, target_train, target_test = get_data(dataset_selec
    for f in feats:
        RBF_PCA_logit_clf.set_params(base_estimator__sampler__n_components =
        RBF_PCA_logit_clf.fit(data_train, target_train)
        train_score = RBF_PCA_logit_clf.score(data_train, target_train)
        test_score = RBF_PCA_logit_clf.score(data_test, target_test)

        RBF_PCA_logit_train_scores.append(train_score)
        RBF_PCA_logit_test_scores.append(test_score)

        sub_progress_bar.value += 1


    train_dic = {
        'absi': feats,
        'ord': RBF_PCA_logit_train_scores,
        'label': 'RBF_PCA_Logit (Grey Box)Train scores'
    }

    test_dic = {
        'absi': feats,
        'ord': RBF_PCA_logit_test_scores,
        'label': 'RBF_PCA_Logit (Grey Box) Test scores'
    }
    return train_dic, test_dic
```

In [17]:

```
def nystroem_pca_rf_greybox_graph():
    sub_clf = Pipeline([
        ("sampler",Nystroem(gamma=0.2)),
        ("pca", PCA(n_components = 0.9, svd_solver = "full")),
        ("clf", DecisionTreeClassifier())
    ])

    NYS_PCA_RF_clf = BaggingClassifier(
        base_estimator = sub_clf,
        n_estimators = n_estimators_selector.value,
        bootstrap = True)


    NYS_PCA_RF_train_scores = []
    NYS_PCA_RF_test_scores = []
    data_train, data_test, target_train, target_test = get_data(dataset_selec

    sub_progress_bar.min = 0
    sub_progress_bar.max = len(feats)
    sub_progress_bar.value = 0

    for f in feats:
        NYS_PCA_RF_clf.set_params(base_estimator__sampler__n_components = f)
        NYS_PCA_RF_clf.fit(data_train, target_train)
        train_score = NYS_PCA_RF_clf.score(data_train, target_train)
        test_score = NYS_PCA_RF_clf.score(data_test, target_test)

        NYS_PCA_RF_train_scores.append(train_score)
        NYS_PCA_RF_test_scores.append(test_score)

        sub_progress_bar.value += 1


    train_dic = {
        'absi': feats,
        'ord': NYS_PCA_RF_train_scores,
        'label': 'NYS_PCA_RF (Grey Box)Train scores'
    }

    test_dic = {
        'absi': feats,
        'ord': NYS_PCA_RF_test_scores,
        'label': 'NYS_PCA_RF (Grey Box) Test scores'
    }
    return train_dic, test_dic
```

In [18]:
```
def rbf_pca_rf_greybox_graph():
    sub_clf = Pipeline([
        ("sampler",RBFSampler(gamma = 0.2)),
        ("pca", PCA(n_components = 0.9, svd_solver = "full")),
        ("clf", DecisionTreeClassifier())
    ])

    RBF_PCA_RF_clf = BaggingClassifier(
        base_estimator = sub_clf,
        n_estimators = n_estimators_selector.value,
        bootstrap = True)


    RBF_PCA_RF_train_scores = []
    RBF_PCA_RF_test_scores = []
    data_train, data_test, target_train, target_test = get_data(dataset_selec

    sub_progress_bar.min = 0
    sub_progress_bar.max = len(feats)
    sub_progress_bar.value = 0

    for f in feats:
        RBF_PCA_RF_clf.set_params(base_estimator__sampler__n_components = f)
        RBF_PCA_RF_clf.fit(data_train, target_train)
        train_score = RBF_PCA_RF_clf.score(data_train, target_train)
        test_score = RBF_PCA_RF_clf.score(data_test, target_test)

        RBF_PCA_RF_train_scores.append(train_score)
        RBF_PCA_RF_test_scores.append(test_score)

        sub_progress_bar.value += 1


    train_dic = {
        'absi': feats,
        'ord': RBF_PCA_RF_train_scores,
        'label': 'RBF_PCA_RF (Grey Box)Train scores'
    }

    test_dic = {
        'absi': feats,
        'ord': RBF_PCA_RF_test_scores,
        'label': 'RBF_PCA_RF (Grey Box) Test scores'
    }
    return train_dic, test_dic
```

In [19]:
```
def show_graphs(e):
    if clear_output_button.value:
        clear_output()
        display(gui)
    display(md("Using dataset **" + str(dataset_selector.value) + "** with **
    global feats
    feats = list(range(*(features_selector.value), features_selector.step))
    feats2 = np.linspace(*(features_selector.value), dtype = np.int64)
    if len(feats2) <= len(feats):
        feats = feats2
    train_dicts = []
    test_dicts = []

    progress_bar.min = 0
    progress_bar.max = sum([i.value for i in ch_boxes])
    progress_bar.value = 0

    for i in range(len(ch_boxes)):
        if ch_boxes[i].value:
            train_dic, test_dic = models[i]()
            train_dicts.append(train_dic)
            test_dicts.append(test_dic)
            progress_bar.value += 1

    fig = plt.figure(figsize=(20,5))
    test_graph = fig.add_subplot(121)
    train_graph = fig.add_subplot(122, sharey = test_graph)

    for i in test_dicts:
        test_graph.plot(i['absi'], i['ord'], label = i['label'])
    for i in train_dicts:
        train_graph.plot(i['absi'], i['ord'], label = i['label'])

    tit = ("Using dataset **" +
          str(dataset_selector.value) +
          "** with **" +
        str(n_estimators_selector.value) +
        "** estimators")
    plt.title(tit)
    train_graph.grid(True)
    test_graph.grid(True)
    train_graph.legend()
    test_graph.legend()
    plt.xlabel("Number of features")
    plt.ylabel("Accuracy")
```

In [20]:
```
def show_graphs_deprecated(e):
    if clear_output_button.value:
        clear_output()
        display(gui)
    display(md("Using dataset **" + str(dataset_selector.value) + "** with **
    global feats
    feats = list(range(*(features_selector.value), features_selector.step))
    feats2 = np.linspace(*(features_selector.value), dtype = np.int64)
    if len(feats2) <= len(feats):
        feats = feats2
    train_dicts = []
    test_dicts = []
    # Random Forest
    if rf_checkbox.value:
        RF_train_dic, RF_test_dic = rf_graph()
        train_dicts.append(RF_train_dic)
        test_dicts.append(RF_test_dic)

    # Random Forest
    if rbf_pca_rf_blackbox_checkbox.value:
        RBF_PCA_RF_train_dic, RBF_PCA_RF_test_dic = rbf_pca_rf_blackbox_graph
        train_dicts.append(RBF_PCA_RF_train_dic)
        test_dicts.append(RBF_PCA_RF_test_dic)

    if rbf_pca_rf_greybox_checkbox.value:
        RBF_PCA_RF_train_dic, RBF_PCA_RF_test_dic = rbf_pca_rf_greybox_graph(
        train_dicts.append(RBF_PCA_RF_train_dic)
        test_dicts.append(RBF_PCA_RF_test_dic)

    # Decision Tree
    if rbf_pca_dt_checkbox.value:
        RBF_PCA_DT_train_dic, RBF_PCA_DT_test_dic = rbf_pca_dt_graph()
        train_dicts.append(RBF_PCA_DT_train_dic)
        test_dicts.append(RBF_PCA_DT_test_dic)

    # Decision Tree
    if dt_checkbox.value:
        DT_train_dic, DT_test_dic = dt_graph()
        train_dicts.append(DT_train_dic)
        test_dicts.append(DT_test_dic)

    if linearSVM_checkbox.value:
        LinearSVM_train_dic, LinearSVM_test_dic = linearSVM_graph()
        train_dicts.append(LinearSVM_train_dic)
        test_dicts.append(LinearSVM_test_dic)
    if logit_checkbox.value:
        logit_train_dic, logit_test_dic = logit_graph()
        train_dicts.append(logit_train_dic)
        test_dicts.append(logit_test_dic)
    if rbf_pca_logit_greybox_checkbox.value:
        rbf_pca_logit_greybox_train_dic, rbf_pca_logit_greybox_test_dic = rbf
        train_dicts.append(rbf_pca_logit_greybox_train_dic)
        test_dicts.append(rbf_pca_logit_greybox_test_dic)
    if rbf_pca_linearSVM_greybox_checkbox.value:
        rbf_pca_linearSVM_greybox_train_dic, rbf_pca_linearSVM_greybox_test_d
        train_dicts.append(rbf_pca_linearSVM_greybox_train_dic)
        test_dicts.append(rbf_pca_linearSVM_greybox_test_dic)
    if nystroem_pca_rf_blackbox_checkbox.value:
        nys_pca_rf_blackbox_train_dic, nys_pca_rf_blackbox_test_dic = nystroe
        train_dicts.append(nys_pca_rf_blackbox_train_dic)
        test_dicts.append(nys_pca_rf_blackbox_test_dic)









    fig = plt.figure(figsize=(20,5))
```

In [21]:
```
bt = widgets.Button(
    description='Show graphs',
    disabled=False,
    button_style='success', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Run the specified methods and show results',
    #icon='check'
)
bt.on_click(show_graphs)

dataset_selector = widgets.Dropdown(
    options=['digits', 'fall_detection','full_covertype','subset_covertype'],
    description='Dataset:'
)
```

In [22]:
```
size_selector = widgets.RadioButtons(
    options={'Small (2000)':2000, 'Medium (4900)':4900, 'Large (10000)':10000
    value=2000,
    #description='Pizza topping:',
    disabled=False,
    orientation = 'horizontal'
)

cool_size_selector = widgets.VBox([widgets.Label("Size of the dataset"), size
```

In [ ]:

In [23]:
```
dt_checkbox = widgets.Checkbox(
    value=True,
    description='DecisionTree',
)

linearSVM_checkbox = widgets.Checkbox(
    value = True,
    description = 'Linear SVM'
)

rf_checkbox = widgets.Checkbox(
    value=True,
    description='Random Forest',
)



rbf_pca_dt_checkbox = widgets.Checkbox(
    value=True,
    description='RBF >> PCA >> DT',
)

rbf_pca_rf_blackbox_checkbox = widgets.Checkbox(
    value=True,
    description='RBF >> PCA >> RF (Black Box Model)',
    layout={'width': '500px'}
)

rbf_pca_rf_greybox_checkbox = widgets.Checkbox(
    value=True,
    description='RBF >> PCA >> RF (Grey Box Model)',
    layout={'width': '500px'}
)

logit_checkbox = widgets.Checkbox(
    value=True,
    description='Logit',
    #layout={'width': '500px'}
)

rbf_pca_logit_greybox_checkbox = widgets.Checkbox(
    value=True,
    #description='RBF >> PCA >> Logit (Grey Box Model)',
    description='RBF >> PCA >> Logit (Ensemble)',
    layout={'width': '500px'}
)


rbf_pca_linearSVM_greybox_checkbox = widgets.Checkbox(
    value=True,
    #description='RBF >> PCA >> LinearSVM (Grey Box Model)',
    description='RBF >> PCA >> LinearSVM (Ensemble)',
    layout={'width': '500px'}
)

nystroem_pca_rf_blackbox_checkbox = widgets.Checkbox(
    value=True,
    #description='RBF >> PCA >> LinearSVM (Grey Box Model)',
    description='Nystroem >> PCA >> RF (Back Box Model)',
    layout={'width': '500px'}
)

nystroem_pca_rf_greybox_checkbox = widgets.Checkbox(
    value=True,
    #description='RBF >> PCA >> LinearSVM (Grey Box Model)',
    description='Nystroem >> PCA >> RF (Grey Box Model)',
    layout={'width': '500px'}
)
```

In [24]:
```
models = [
    dt_graph,
    linearSVM_graph,
    rbf_pca_linearSVM_greybox_graph,
    logit_graph,
    rbf_pca_logit_greybox_graph,
    rf_graph,
    rbf_pca_dt_graph,
    rbf_pca_rf_blackbox_graph,
    rbf_pca_rf_greybox_graph,
    nystroem_pca_rf_blackbox_graph,
    nystroem_pca_rf_greybox_graph,
    nystroem_pca_linearSVM_greybox_graph,
    nystroem_pca_logit_greybox_graph,
    nystroem_pca_dt_graph,
]
'''
        dt_graph
        linearSVM_graph
        rbf_pca_linearSVM_greybox_graph
        logit_graph
        rbf_pca_logit_greybox_graph
        rf_graph
        rbf_pca_dt_graph
        rbf_pca_rf_blackbox_graph
        rbf_pca_rf_greybox_graph
        nystroem_pca_rf_blackbox_graph
'''



ch_boxes = [
    dt_checkbox,
    linearSVM_checkbox,
    rbf_pca_linearSVM_greybox_checkbox,
    logit_checkbox,
    rbf_pca_logit_greybox_checkbox,
    rf_checkbox,
    rbf_pca_dt_checkbox,
    rbf_pca_rf_blackbox_checkbox,
    rbf_pca_rf_greybox_checkbox,
    nystroem_pca_rf_blackbox_checkbox,
    nystroem_pca_rf_greybox_checkbox,
    nystroem_pca_linearSVM_greybox_checkbox,
    nystroem_pca_logit_greybox_checkbox,
    nystroem_pca_dt_checkbox,
]
```

In [25]:
```
'''
models_selector = widgets.VBox([
    dt_checkbox,
    linearSVM_checkbox,
    rbf_pca_linearSVM_greybox_checkbox,
    logit_checkbox,
    rbf_pca_logit_greybox_checkbox,
    rf_checkbox,
    rbf_pca_dt_checkbox,
    rbf_pca_rf_blackbox_checkbox,
    rbf_pca_rf_greybox_checkbox,
    nystroem_pca_rf_blackbox_checkbox,
    nystroem_pca_rf_greybox_checkbox,
])
'''
#models_selector = widgets.VBox(ch_boxes)
```

Out[25]:
```
'\nmodels_selector = widgets.VBox([\n    dt_checkbox,\n    linearSVM_checkbox,
\n    rbf_pca_linearSVM_greybox_checkbox,\n    logit_checkbox,\n    rbf_pca_lo
git_greybox_checkbox,\n    rf_checkbox,\n    rbf_pca_dt_checkbox,\n    rbf_pca
_rf_blackbox_checkbox,\n    rbf_pca_rf_greybox_checkbox,\n    nystroem_pca_rf_
blackbox_checkbox,\n    nystroem_pca_rf_greybox_checkbox,\n])\n'
```

In [26]:
```
rbf_models_selector = widgets.VBox([
    rbf_pca_linearSVM_greybox_checkbox,
    rbf_pca_logit_greybox_checkbox,
    rbf_pca_dt_checkbox,
    rbf_pca_rf_greybox_checkbox,
    rbf_pca_rf_blackbox_checkbox,
])

nystroem_models_selector = widgets.VBox([
    nystroem_pca_linearSVM_greybox_checkbox,
    nystroem_pca_logit_greybox_checkbox,
    nystroem_pca_dt_checkbox,
    nystroem_pca_rf_greybox_checkbox,
    nystroem_pca_rf_blackbox_checkbox,
])

normal_models_selector = widgets.VBox([
    linearSVM_checkbox,
    logit_checkbox,
    dt_checkbox,
    rf_checkbox,
])

models_selector = widgets.HBox([
    rbf_models_selector,
    nystroem_models_selector,
    normal_models_selector,
])
```

In [27]:
```
n_estimators_selector = widgets.IntSlider(
    value=10,
    min=10,
    max=200,
    step=5,
    #description='Number of estimators:',
    orientation='horizontal',
    #readout=True,
    #readout_format='d'
)

cool_nest_selector = widgets.HBox([widgets.Label("Number of estimators: "), n
```

In [28]:
```
clear_output_button = widgets.ToggleButton(
    value=True,
    description='Clear Previous',
    button_style='', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Clear Previous Output',
    icon='check'
)
```

In [29]:
```
features_selector = widgets.IntRangeSlider(
    value=[30, 150],
    min=30,
    max=500,
    step=10,
    #description='Number of features:',
    #continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='d',
)

cool_features_selector = widgets.HBox([widgets.Label("Range of features: "),
```

In [30]:
```
def deselect_all(b):
    for i in ch_boxes:
        i.value = False
    #for i in models_selector.children:
     #   i.value = False

def select_all(b):
    for i in ch_boxes:
        i.value = True
    #for i in models_selector.children:
     #   i.value = True
```

In [31]:
```
select_all_button = widgets.Button(
    description='Select All',
    disabled=False,
    button_style='info', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Select all models',
    #icon='check'
)
select_all_button.on_click(select_all)
```

In [32]:
```
deselect_all_button = widgets.Button(
    description='Deselect All',
    disabled=False,
    button_style='info', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Select all models',
    #icon='check'
)
deselect_all_button.on_click(deselect_all)
```

In [33]:
```
progress_bar = widgets.IntProgress(
    value=0,
    min=0,
    max=10,
    step=1,
    description='Calculating:',
    bar_style='info', # 'success', 'info', 'warning', 'danger' or ''
    orientation='horizontal'
)
```

In [34]:
```
sub_progress_bar = widgets.IntProgress(
    value=0,
    min=0,
    max=10,
    step=1,
    #description='',
    bar_style='info', # 'success', 'info', 'warning', 'danger' or ''
    orientation='horizontal'
)
```

In [35]:
```
gui = widgets.VBox([
    cool_size_selector,
    dataset_selector,
    models_selector,
    select_all_button,
    deselect_all_button,
    cool_nest_selector,
    cool_features_selector,
    bt,
    clear_output_button,
    progress_bar,
    sub_progress_bar,
])
```

In [36]:
```
n_estimators_selector.value = 100
```

In [37]:
```
display(gui)
```

Size of the dataset

- (•) Small (2000)
- ( ) Medium (4900)
- ( ) Large (10000)
- ( ) Full Dataset

Dataset: subset_covertype

[ ] RBF >> PCA >> Linear	[ ] Nystroem >> PCA >> Li	[ ] Linear
[ ] RBF >> PCA >> Logit (I	[ ] Nystroem >> PCA >> L	[ ] Logit
[x] RBF >> PCA >> DT	[x] Nystroem >> PCA >> D	[x] Decisi
[ ] RBF >> PCA >> RF (Gr	[ ] Nystroem >> PCA >> R	[ ] Rando
[ ] RBF >> PCA >> RF (Bl	[ ] Nystroem >> PCA >> R	

Select All

Deselect All

Number of estimators: 30

Range of features: 50 – 150

Show graphs

✔ Clear Previous

Calculating:

Vienen ahora una demos para comprobar cosas específicas

In [38]:
```
# Acciones:
# Usar los modelos simples para cada uno de los datasets
# Observaciones:
#
def demo1(e):
    size_selector.value = 2000

    deselect_all(None)

    linearSVM_checkbox.value = True
    logit_checkbox.value = True
    dt_checkbox.value = True
    rf_checkbox.value = True

    n_estimators_selector.value = 100

    clear_output_button.value = False

    #datasets = ['digits', 'fall_detection','subset_covertype', "full_coverty
    datasets = ['digits', 'fall_detection','subset_covertype']
    for d in datasets:
        dataset_selector.value = d
        show_graphs(None)
```

In [39]:
```
# Acciones:
# Todos los modelos que usan RBF con cada uno de los datasets
#
#
def demo2(e):
    size_selector.value = 2000

    deselect_all(None)

    rbf_pca_dt_checkbox.value = True
    rbf_pca_rf_blackbox_checkbox.value = True
    rbf_pca_rf_greybox_checkbox.value = True
    rbf_pca_logit_greybox_checkbox.value = True
    rbf_pca_linearSVM_greybox_checkbox.value = True


    n_estimators_selector.value = 30

    features_selector.min = 50
    features_selector.max = 200
    features_selector.step = 10

    clear_output_button.value = False

    #datasets = ['digits', 'fall_detection','subset_covertype', "full_coverty
    datasets = ['digits', 'fall_detection','subset_covertype']
    for d in datasets:
        dataset_selector.value = d
        show_graphs(None)
```

In [40]:
```
# Acciones:
# Todos los modelos que usan Nystroem con cada uno de los datasets
#
#
def demo3(e):
    size_selector.value = 2000

    deselect_all(None)


    nystroem_pca_rf_blackbox_checkbox.value = True
    nystroem_pca_rf_greybox_checkbox.value = True
    nystroem_pca_linearSVM_greybox_checkbox.value = True
    nystroem_pca_logit_greybox_checkbox.value = True
    nystroem_pca_dt_checkbox.value = True


    n_estimators_selector.value = 30

    features_selector.min = 50
    features_selector.max = 200
    features_selector.step = 10

    clear_output_button.value = False

    #datasets = ['digits', 'fall_detection','subset_covertype', "full_coverty
    datasets = ['digits', 'fall_detection','subset_covertype']
    for d in datasets:
        dataset_selector.value = d
        show_graphs(None)
```

In [41]:
```
# currentwork
# Acciones:
# Todos los modelos de DT, que son 4, con cada uno de los datasets
#
#
def demo4(e):
    size_selector.value = 2000

    deselect_all(None)


    dt_checkbox.value = True
    rbf_pca_dt_checkbox.value = True
    nystroem_pca_dt_checkbox.value = True


    n_estimators_selector.value = 30

    features_selector.min = 50
    features_selector.max = 200
    features_selector.step = 10

    clear_output_button.value = False

    #datasets = ['digits', 'fall_detection','subset_covertype', "full_coverty
    datasets = ['digits', 'fall_detection','subset_covertype']
    for d in datasets:
        dataset_selector.value = d
        show_graphs(None)
```

In [42]:
```
def run_all_demos(e):
    for i in range(1,5):
        exec('demo{0}(None)'.format(i))
```

In [43]:
```
demos_str = '''bt_demo{0} = widgets.Button(
    description='Demo {0}',
    #disabled=False,
    button_style='warning', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Run Demo{0}',
    #icon='check'
)
bt_demo{0}.on_click(demo{0})
'''
all_demos_bt = widgets.Button(
    description = 'Run all demos',
    button_style = 'success',
    tooltip='Run all demos',
)

all_demos_bt.on_click(run_all_demos)
```

In [45]:
```
display(bt_demo1)
```

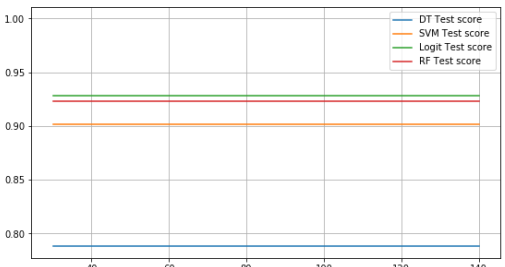
Demo 1

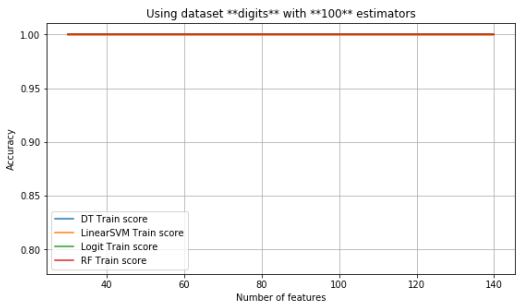
Using dataset **digits** with **100** estimators

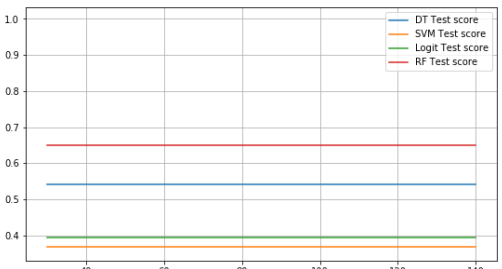
Using dataset **fall_detection** with **100** estimators

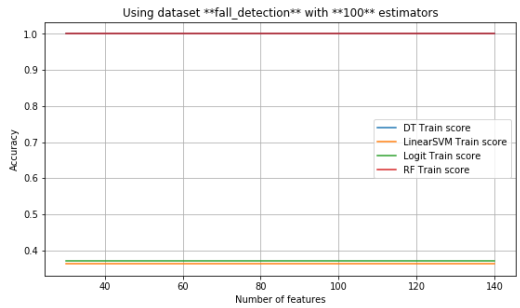
Using dataset **subset_covertype** with **100** estimators

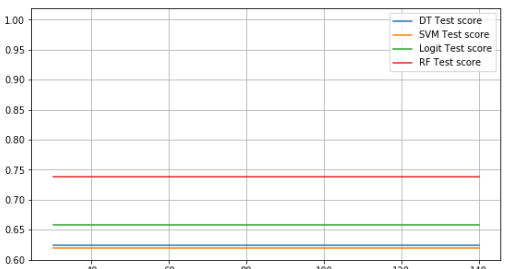
```
/home/hobber/.local/lib/python3.6/site-packages/sklearn/linear_model/logistic.
py:757: ConvergenceWarning: lbfgs failed to converge. Increase the number of i
terations.
  "of iterations.", ConvergenceWarning)
```

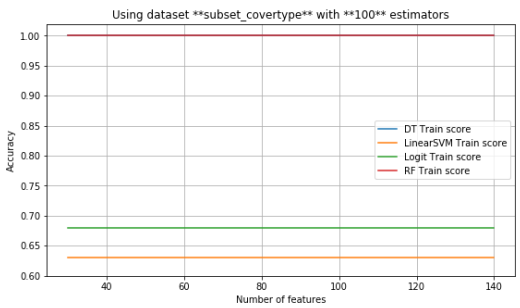












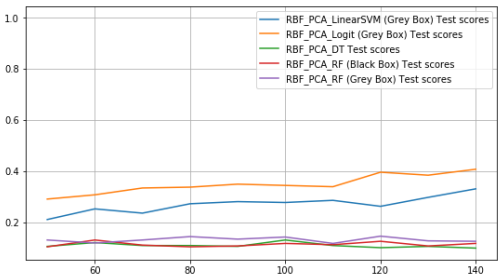
In [46]:
```
display(bt_demo2)
```

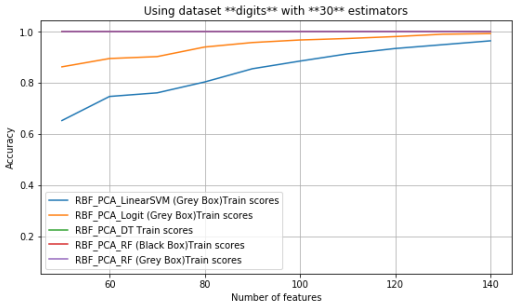
Demo 2

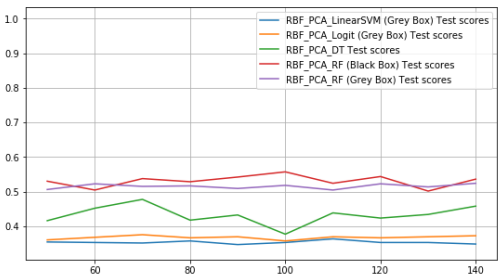
Using dataset **digits** with **30** estimators

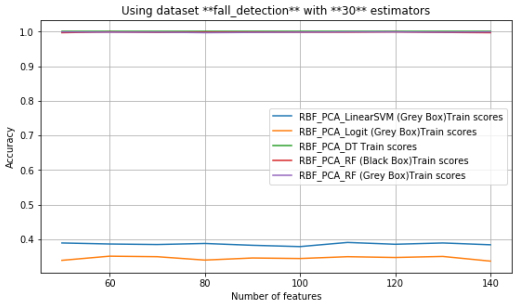
Using dataset **fall_detection** with **30** estimators

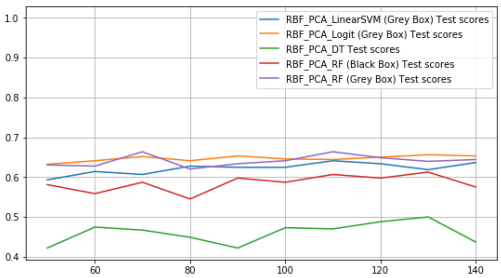
Using dataset **subset_covertype** with **30** estimators

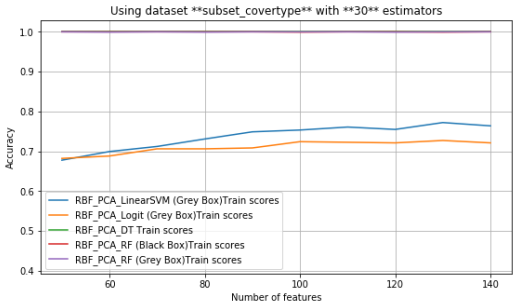












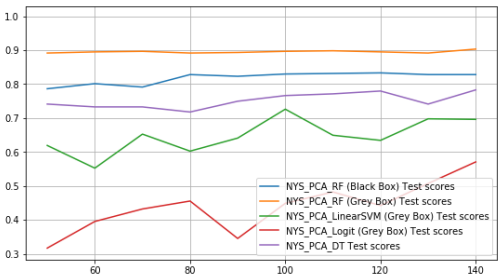
In [47]:
```
display(bt_demo3)
```

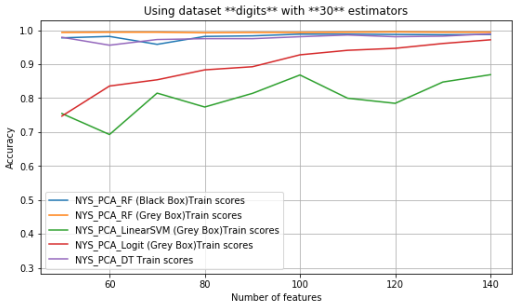
Demo 3

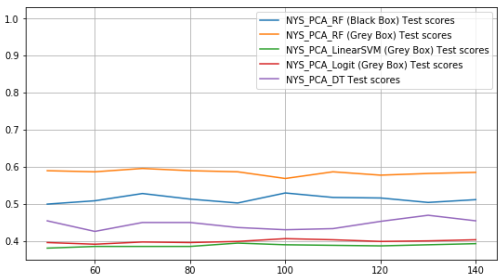
Using dataset **digits** with **30** estimators

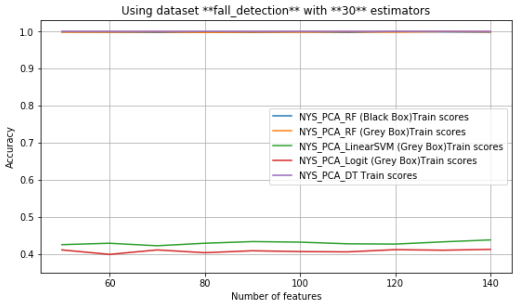
Using dataset **fall_detection** with **30** estimators

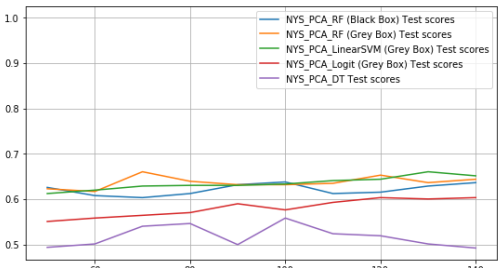
Using dataset **subset_covertype** with **30** estimators

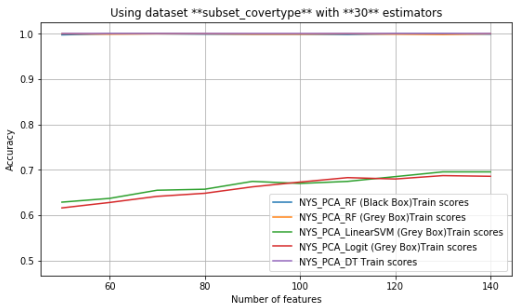












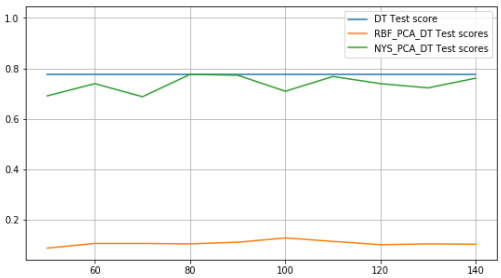
In [48]:
```
display(bt_demo4)
```

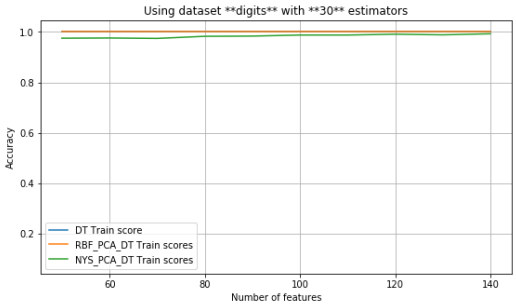
Demo 4

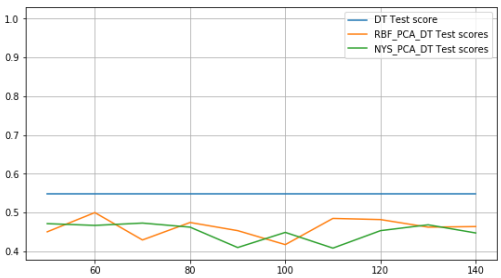
Using dataset **digits** with **30** estimators

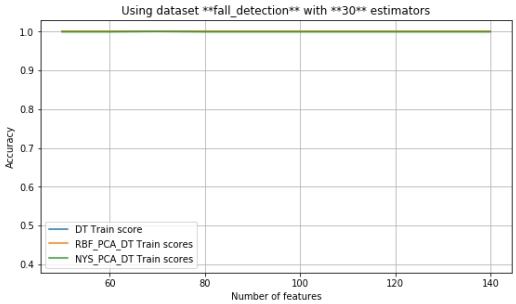
Using dataset **fall_detection** with **30** estimators

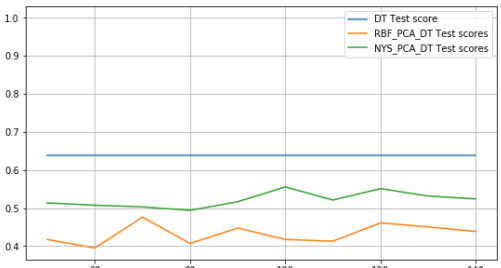
Using dataset **subset_covertype** with **30** estimators

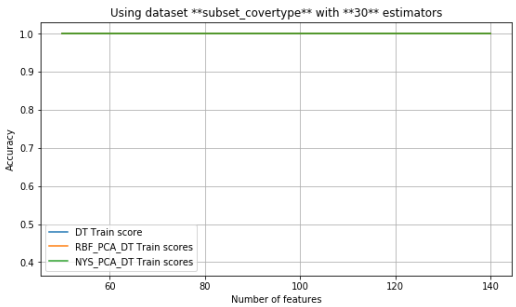












In [44]:
```
for i in range(1,5):
    exec(demos_str.format(i))

for i in range(1,5):
    exec('display(bt_demo{0})'.format(i))
display(all_demos_bt)
display(progress_bar)
display(sub_progress_bar)
```

Demo 1

Demo 2

Demo 3

Demo 4

Run all demos

Calculating: