MIDTERM PROJECT SCRIPT FOR MULTI-CLASS CLASSIFICATION USING FOUR SUPERVISED LEARNING MODELS FROM SCIKIT-LEARN NAMELY PERCEPTRON, LOGISTIC REGRESSION, SVM AND KNN

Python imports

In [104]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap
import csv
from sklearn import datasets
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

Loading the generated dataset using script *dataGen_Script.py* from *midterm_data.csv* file

In [105]:

```
#Loading into a dataframe df
df = pd.read_csv('midterm_data.csv')
print(df.tail())
print("Dimensions of data: ",df.shape)
```

```
           X1        X2  Class
295  1.484537  6.443709      3
296  2.302164  6.158445      3
297  1.115258  6.945472      3
298  1.902292  6.298797      3
299  1.616267  6.310493      3
Dimensions of data:  (300, 3)
```

We see that the dataset is labelled and has 300 samples and 3 columns of which 2 are the features X1 and X2 columns and the last one is the class label.

Extracting all the samples from the above dataframe *df* and the 2 features X1 and X2 (or column names) into X and class labels (third column) into y

In [106]:

```
X = df.iloc[:, [0,1]].values
y = df.iloc[:,2].values
print(y)

#Check the unique class labels to prepare for classification
print('Class labels:', np.unique(y))
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3]
Class labels: [1 2 3]
```

And the samples belong to the class labels that are of 3 types or 3 numbers: 1, 2 and 3. X variable has X1 and X2 columns, y variable has class labels for all the 300 samples.

Splitting the dataset into training and test datasets

In [107]:

```
#random state is for initial shuffling of training dataset after each epoch
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=0)
```

Splitted data into 70% training and 30% test data.

Feature Scaling for Optimal Performance

In [108]:

```
#Standardizing the features
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

#Stack arrays so as to make a single array vertically and horizontally
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
```

Plotting function for decision regions in the Classification Problem

In [109]:

```
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor='black')

    # highlight test samples
    if test_idx:
        # plot all samples
        X_test, y_test = X[test_idx, :], y[test_idx]

        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
                    c='',
                    edgecolor='black',
                    alpha=1.0,
                    linewidth=1,
                    marker='o',
                    s=100,
                    label='test set')
```

Model I --> Perceptron: Training a perceptron using Scikit-learn and Plotting with Decision Regions and Classified Data by specifying the indices of samples (test data)

In [110]:

```
##Multi-class classification using Scikit-learn perceptron
ppn = Perceptron(max_iter=50, eta0=0.1, random_state=1)
ppn.fit(X_train_std, y_train)
y_pred = ppn.predict(X_test_std)
print(y_pred)
print("Weights assigned to the features: ",ppn.coef_)
print('Number of Misclassified samples: %d' % (y_test != y_pred).sum())
#print('Model Accuracy: %.2f' % accuracy_score(y_test, y_pred))
print('Model Accuracy: %.2f' % ppn.score(X_test_std, y_test))
```

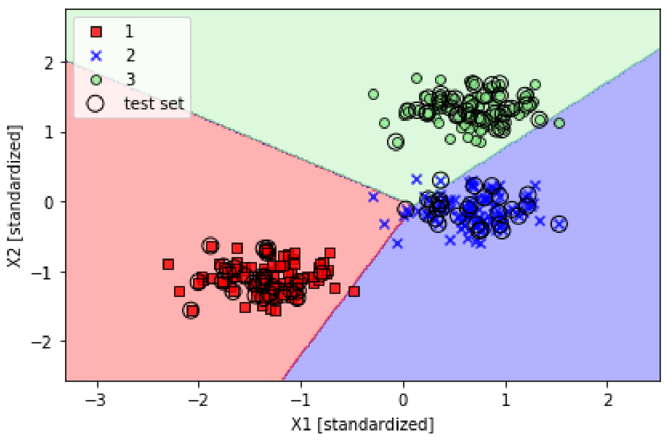
```
[3 2 1 3 3 2 3 3 1 2 3 1 2 3 1 1 3 1 3 2 2 2 2 2 3 1 2 3 3 3 1 1 1 2 3 3 2
 2 3 1 3 1 1 1 1 2 3 1 1 1 1 3 2 2 1 3 2 1 3 2 2 2 1 3 3 3 3 2 1 1 2 3 3 1
 2 3 1 1 2 2 2 2 3 3 3 1 3 3 3 3]
Weights assigned to the features:  [[-0.19368789 -0.03674891]
 [ 0.53036155 -0.41125304]
 [-0.04578494  0.20515548]]
Number of Misclassified samples: 2
Model Accuracy: 0.98
```

Trained the perceptron model using fit method. We could see that the misclassified samples is just 1 and we get the accuracy of 99%. This is an evaluation parameter for the model. I used the same parameter for comparing how the four models perform on our dataset later in this Notebook.

Training the perceptron model using the standardized training data. 30% of test data i.e. last 90 samples are the test data i.e. from 210 to 300 samples. Plot with classification and decision regions is stored in a file called *ppn_classify_plot.png*.

In [111]:

```
plot_decision_regions(X=X_combined_std, y=y_combined,
                      classifier=ppn, test_idx=range(210, 300))
plt.xlabel('X1 [standardized]')
plt.ylabel('X2 [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.savefig('ppn_classify_plot.png', dpi=300)
plt.show()
```



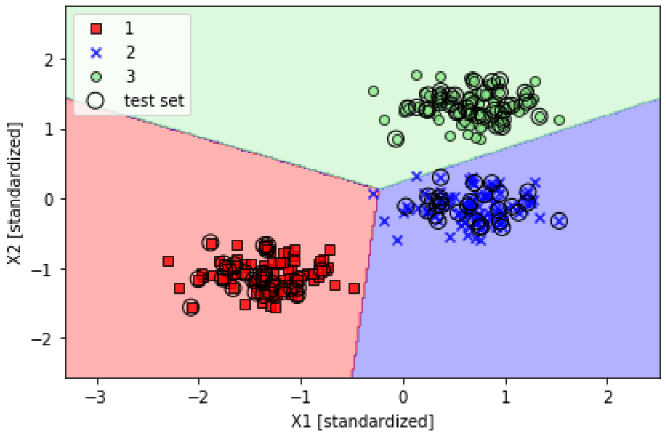
Model II --> Logistic Regression: Training a Logistic Regression model using Scikit-learn and Plotting with Decision Regions and Classified Data by specifying the indices of samples (test

data)

In [112]:

```
lr = LogisticRegression(C=100.0, random_state=1)
lr.fit(X_train_std, y_train)
y_pred_lr = lr.predict(X_test_std)
print('Number of Misclassified samples: %d' % (y_test != y_pred_lr).sum())
#print('Accuracy: %.2f' % accuracy_score(y_test, y_pred_lr))
print('Accuracy: %.2f' % lr.score(X_test_std, y_test))
plot_decision_regions(X_combined_std, y_combined,
                      classifier=lr, test_idx=range(210, 300))
plt.xlabel('X1 [standardized]')
plt.ylabel('X2 [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.savefig('lr_classify_plot.png', dpi=300)
plt.show()
```

```
Number of Misclassified samples: 0
Accuracy: 1.00
```



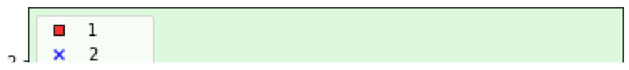
Model III --> Support Vector Machines (SVM): Training a SVM model using Scikit-learn and Plotting with Decision Regions and Classified Data by specifying the indices of samples (test data)

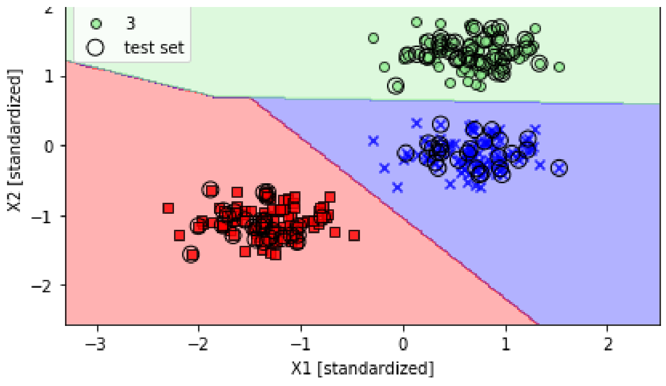
Training SVM model using Scikit-learn and linear kernel.

In [113]:

```
#With linear kernel
svm = SVC(kernel='linear', C=1.0, random_state=1)
svm.fit(X_train_std, y_train)
y_pred_svm = svm.predict(X_test_std)
print('Number of Misclassified samples: %d' % (y_test != y_pred_svm).sum())
#print('Accuracy: %.2f' % accuracy_score(y_test, y_pred_svm))
print('Accuracy: %.2f' % svm.score(X_test_std, y_test))

plot_decision_regions(X_combined_std,
                      y_combined,
                      classifier=svm,
                      test_idx=range(210, 300))
plt.xlabel('X1 [standardized]')
plt.ylabel('X2 [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.savefig('svmlin_classify_plot.png', dpi=300)
plt.show()
```

```
Number of Misclassified samples: 0
Accuracy: 1.00
```



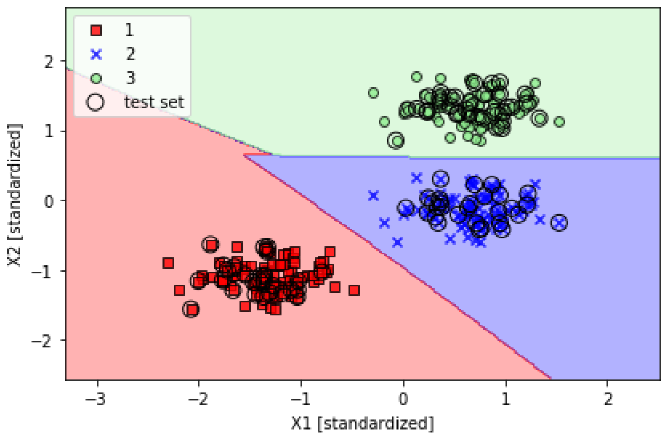


Training SVM model using Scikit-learn and non-linear kernel called Radial Basis Function (RBF) for softer decision boundary.

In [114]:

```
svm_nonlinear = SVC(kernel='rbf',random_state=0,gamma=0.2, C=1.0)
svm_nonlinear.fit(X_train_std, y_train)
y_pred_svm_nl = svm_nonlinear.predict(X_test_std)
print('Number of Misclassified samples: %d' % (y_test != y_pred_svm_nl).sum())
#print('Accuracy: %.2f' % accuracy_score(y_test, y_pred_svm_nl))
print('Accuracy: %.2f' % svm_nonlinear.score(X_test_std, y_test))
plot_decision_regions(X_combined_std,
                      y_combined,
                      classifier=svm_nonlinear,
                      test_idx=range(210, 300))
plt.xlabel('X1 [standardized]')
plt.ylabel('X2 [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.savefig('svmnon_lin_classify_plot.png', dpi=300)
plt.show()
```

```
Number of Misclassified samples: 0
Accuracy: 1.00
```



Model IV --> K-Nearest Neighbours (KNN): Training a KNN model using Scikit-learn and Plotting with Decision Regions and Classified Data by specifying the indices of samples (test data)

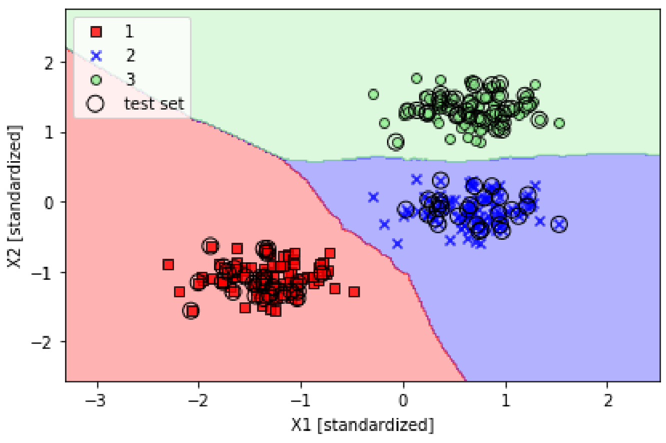
with p=2 i.e. Euclidean distance metric

In [115]:

```
knn = KNeighborsClassifier(n_neighbors=5,
                           p=2,
                           metric='minkowski')
knn.fit(X_train_std, y_train)

y_pred_knn = knn.predict(X_test_std)
print('Number of Misclassified samples: %d' % (y_test != y_pred_knn).sum())
#print('Accuracy: %.2f' % accuracy_score(y_test, y_pred_knn))
```

```
print('Accuracy: %.2f' % knn.score(X_test_std, y_test))

plot_decision_regions(X_combined_std, y_combined,
                      classifier=knn, test_idx=range(210, 300))

plt.xlabel('X1 [standardized]')
plt.ylabel('X2 [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.savefig('knn_classify_plot.png', dpi=300)
plt.show()
```

```
Number of Misclassified samples: 0
Accuracy: 1.00
```



Summary of the models with misclassified samples and accuracy (in %)

Table below lists the four trained models on our dataset with number of misclassified samples and accuracy.

Model	Misclassifications	Accuracy
Perceptron	1	99%
Logistic Regression	1	99%
SVM	0	100%
KNN	0	100%

Comparing the four models

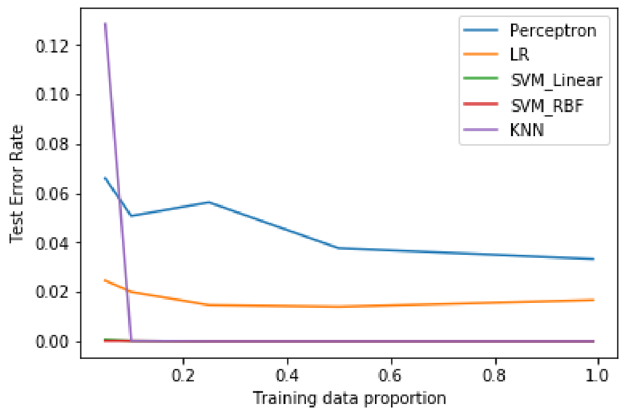
Script and plot showing how the four models (perceptron, logistic regression, svm linear and non-linear and knn) perform on our *midterm_data.csv* dataset.

In [116]:

```
#Proportion of samples
heldout = [0.95, 0.90, 0.75, 0.50, 0.01]
rounds = 20

#Classifiers (supervised learning models used in this analysis) as a list and initialized
classifiers = [
    ("Perceptron", Perceptron(max_iter=50, eta0=0.1, random_state=1)),
    ("LR", LogisticRegression(C=100.0, random_state=1)),
    ("SVM_Linear",SVC(kernel='linear', C=1.0, random_state=1)),
    ("SVM_RBF",SVC(kernel='rbf',random_state=0,gamma=0.2, C=1.0)),
    ("KNN",KNeighborsClassifier(n_neighbors=5,
                              p=2,
                              metric='minkowski'))
]

#Training sample size proportion
xx = 1. - np.array(heldout)
```

```
#Computing the score below by training the models and predicting the class labels and classifying
them by looping through the list of classifiers declared above
for name, clf in classifiers:
    print("training %s" % name)
    rng = np.random.RandomState(42)
    yy = []
    for i in heldout:
        yy_ = []
        for r in range(rounds):
            X_train, X_test, y_train, y_test = \
                train_test_split(X, y, test_size=i, random_state=rng)
            clf.fit(X_train, y_train)
            y_pred = clf.predict(X_test)
            yy_.append(1 - np.mean(y_pred == y_test))
        yy.append(np.mean(yy_))
    plt.plot(xx, yy, label=name)

plt.legend(loc="upper right")
plt.xlabel("Training data proportion")
plt.ylabel("Test Error Rate")
plt.show()
```

```
training Perceptron
training LR
training SVM_Linear
training SVM_RBF
training KNN
```



From the above analysis we see that the test error rate for **perceptron** model initially shows few misclassifications but reduces it as the training data proportion increases. Whereas the **Logistic Regression** model almost stays constant at low test error rate for any training set proportion at 0.02. **SVM Linear** and **SVM RBF** have 0 test error rate proving to be the best models for our current dataset. **KNN** model follows the same trend as **Logistic Regression** model but starting with higher test error rate and sudden reduction to 0 test error rate with the increase in training dataset. This shows and conforms to the fact that KNN remembers the dataset from the point it stops misclassifying.

From the classification plot of the four models with decision regions, we see that except **Perceptron** and **Logistic Regression**, **SVM** and **KNN** show the clear classification of samples based on their class labels.