Notebook 1

Exploração de Dados 2018/2019

Nelson Costa 42983

Armando Sousa 76498

Index

In [1]:

```
import pandas as pd
import numpy as np
import xlrd
import copy
%matplotlib inline
```

In [2]:

```
#Disable warning
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Reading datasets

Notebook 1 is divided into 3 parts: Pre processing and predictive model. The datasets used are 'Banknote Authentication Data Set' (https://archive.ics.uci.edu/ml/datasets/banknote+authentication (https://archive.ics.uci.edu/ml/datasets/banknote+authentication)) and 'Nursery Data Set' (https://archive.ics.uci.edu/ml/datasets/Nursery (https://archive.ics.uci.edu/ml/datasets/Nursery)).

Banknotes dataset

In [3]:

```
df_bankNotes = pd.read_excel('./data_banknote_authentication.xlsx')
#df_bankNotes
```

Nursery dataset

In [4]:

```
df_nursery = pd.read_excel('./nursery.xlsx')
#df_nursery
```

Task A - Pre Processing

Checking if datasets have missing values

In this case none of the datasets have missing values,

Banknotes dataset

In [5]:

```
df_bankNotes.isnull().sum()
```

Out[5]:

```
variance of Wavelet Transformed image    0
skewness of Wavelet Transformed image    0
curtosis of Wavelet Transformed image    0
entropy of image                         0
class                                    0
dtype: int64
```

Nursery dataset

In [6]:

```
df_nursery.isnull().sum()
```

Out[6]:

```
parents     0
has_nurs    0
form        0
children    0
housing     0
finance     0
social      0
health      0
class       0
dtype: int64
```

Handling categorical data

Mapping ordinal features

Nursery dataset

In [7]:

```
# print features before mapping
#df_nursery[df_nursery.columns[0:8]]
```

In [8]:

```
# get dataset copy
df_nursery_copy = df_nursery.copy()

# cast data to string
df_nursery_copy = df_nursery_copy.astype(str)

# map features
parents_mapping = {'usual':1, 'pretentious':2, 'great_pret':3}
df_nursery_copy['parents'] = df_nursery_copy['parents'].map(parents_mapping)

has_nurs_mapping = {'proper':1, 'less_proper':2, 'improper':3, 'critical':4, 've
ry_crit':5}
df_nursery_copy['has_nurs'] = df_nursery_copy['has_nurs'].map(has_nurs_mapping)

form_mapping = {'complete':1, 'completed':2, 'incomplete':3, 'foster':4}
df_nursery_copy['form'] = df_nursery_copy['form'].map(form_mapping)

children_mapping = {'1':1, '2':2, '3':3, 'more':4}
df_nursery_copy['children'] = df_nursery_copy['children'].map(children_mapping)

housing_mapping = {'convenient':1, 'less_conv':2, 'critical':3}
df_nursery_copy['housing'] = df_nursery_copy['housing'].map(housing_mapping)

finance_mapping = {'convenient':1, 'inconv':2}
df_nursery_copy['finance'] = df_nursery_copy['finance'].map(finance_mapping)

social_mapping = {'nonprob':1, 'slightly_prob':2, 'problematic':3}
df_nursery_copy['social'] = df_nursery_copy['social'].map(social_mapping)

health_mapping = {'recommended':1, 'priority':2, 'not_recom':3}
df_nursery_copy['health'] = df_nursery_copy['health'].map(health_mapping)
```

In [9]:

```
# print features after mapping
df_nursery_copy[df_nursery_copy.columns[0:8]]
```

Out[9]:

	parents	has_nurs	form	children	housing	finance	social	health
0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	2
2	1	1	1	1	1	1	1	3
3	1	1	1	1	1	1	2	1
4	1	1	1	1	1	1	2	2
5	1	1	1	1	1	1	2	3
6	1	1	1	1	1	1	3	1
7	1	1	1	1	1	1	3	2
8	1	1	1	1	1	1	3	3
9	1	1	1	1	1	2	1	1
10	1	1	1	1	1	2	1	2
11	1	1	1	1	1	2	1	3
12	1	1	1	1	1	2	2	1
13	1	1	1	1	1	2	2	2
14	1	1	1	1	1	2	2	3
15	1	1	1	1	1	2	3	1
16	1	1	1	1	1	2	3	2
17	1	1	1	1	1	2	3	3
18	1	1	1	1	2	1	1	1
19	1	1	1	1	2	1	1	2
20	1	1	1	1	2	1	1	3
21	1	1	1	1	2	1	2	1
22	1	1	1	1	2	1	2	2
23	1	1	1	1	2	1	2	3
24	1	1	1	1	2	1	3	1
25	1	1	1	1	2	1	3	2
26	1	1	1	1	2	1	3	3
27	1	1	1	1	2	2	1	1
28	1	1	1	1	2	2	1	2
29	1	1	1	1	2	2	1	3
...	...	...	...	...	...	...	...	...
12930	3	5	4	4	2	1	3	1
12931	3	5	4	4	2	1	3	2
12932	3	5	4	4	2	1	3	3
12933	3	5	4	4	2	2	1	1
12934	3	5	4	4	2	2	1	2
12935	3	5	4	4	2	2	1	3

	parents	has_nurs	form	children	housing	finance	social	health
12936	3	5	4	4	2	2	2	1
12937	3	5	4	4	2	2	2	2
12938	3	5	4	4	2	2	2	3
12939	3	5	4	4	2	2	3	1
12940	3	5	4	4	2	2	3	2
12941	3	5	4	4	2	2	3	3
12942	3	5	4	4	3	1	1	1
12943	3	5	4	4	3	1	1	2
12944	3	5	4	4	3	1	1	3
12945	3	5	4	4	3	1	2	1
12946	3	5	4	4	3	1	2	2
12947	3	5	4	4	3	1	2	3
12948	3	5	4	4	3	1	3	1
12949	3	5	4	4	3	1	3	2
12950	3	5	4	4	3	1	3	3
12951	3	5	4	4	3	2	1	1
12952	3	5	4	4	3	2	1	2
12953	3	5	4	4	3	2	1	3
12954	3	5	4	4	3	2	2	1
12955	3	5	4	4	3	2	2	2
12956	3	5	4	4	3	2	2	3
12957	3	5	4	4	3	2	3	1
12958	3	5	4	4	3	2	3	2
12959	3	5	4	4	3	2	3	3

12960 rows × 8 columns

Encoding class labels

Nursery dataset

In [10]:

```
# print labels before encoding
#df_nursery_copy[df_nursery_copy.columns[8]]
```

In [11]:

```
class_mapping = {label:idx for idx,label in enumerate(np.unique(df_nursery_copy[
df_nursery_copy.columns[8]]))}

class_mapping
```

Out[11]:

```
{'not_recom': 0,
 'priority': 1,
 'recommend': 2,
 'spec_prior': 3,
 'very_recom': 4}
```

In [12]:

```
# map labels
df_nursery_copy[df_nursery_copy.columns[8]] = df_nursery_copy[df_nursery_copy.co
lumns[8]].map(class_mapping)
```

In [13]:

```
# print labels after encoding
df_nursery_copy[df_nursery_copy.columns[8]]
```

Out[13]:

```
0        2
1        1
2        0
3        2
4        1
5        0
6        1
7        1
8        0
9        4
10       1
11       0
12       4
13       1
14       0
15       1
16       1
17       0
18       4
19       1
20       0
21       4
22       1
23       0
24       1
25       1
26       0
27       4
28       1
29       0
        ..
12930    3
12931    3
12932    0
12933    3
12934    3
12935    0
12936    3
12937    3
12938    0
12939    3
12940    3
12941    0
12942    3
12943    3
12944    0
12945    3
12946    3
12947    0
12948    3
12949    3
12950    0
12951    3
12952    3
12953    0
12954    3
12955    3
12956    0
12957    3
```

```
12958    3
12959    0
Name: class, Length: 12960, dtype: int64
```

Partitioning datasets in training and test sets

In [14]:

```
# Added version check for recent scikit-learn 0.18 checks
from distutils.version import LooseVersion as Version
from sklearn import __version__ as sklearn_version

if Version(sklearn_version) < '0.18':
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split
```

Banknotes dataset

In [15]:

```
# labels reading
y1=df_bankNotes[df_bankNotes.columns[4]]
# features reading
X1=df_bankNotes[df_bankNotes.columns[0:4]]

# get training and test sets
X_train1,X_test1,y_train1,y_test1 = train_test_split(X1,y1,test_size = 0.3)
```

Nursery dataset

In [16]:

```
# labels reading
y2=df_nursery_copy[df_nursery_copy.columns[8]]
# features reading
X2=df_nursery_copy[df_nursery_copy.columns[0:8]]

# get training and test sets
X_train2,X_test2,y_train2,y_test2 = train_test_split(X2,y2,test_size = 0.3)
```

Rank features

Univariate Feature Selection

Univariate feature selection selects the best features by running univariate statistical tests like **chi-squared test, F-1 test, and mutual information** methods. Can't use the **chi-squared** function, if there are negative values.

1. Mutual Info Classif

In [17]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, mutual_info_classif

test = SelectKBest(score_func = mutual_info_classif, k=2)
test
```

Out[17]:

```
SelectKBest(k=2, score_func=<function mutual_info_classif at 0x7f316
2a480d0>)
```

Banknotes dataset

In [18]:

```
test.fit(X_train1, y_train1)
num_features = len(X_train1.columns)

scores = []
for i in range(num_features):
    score = test.scores_[i]
    scores.append((score, X_train1.columns[i]))

print (sorted(scores, reverse = True))
```

```
[(0.3651135027748482, 'variance of Wavelet Transformed image'), (0.2
3575549040416321, 'skewness of Wavelet Transformed image'), (0.12651
335887606008, 'curtosis of Wavelet Transformed image'), (0.016044778
28245643, 'entropy of image')]
```

Nursery dataset

In [19]:

```
test.fit(X_train2, y_train2)
num_features = len(X_train2.columns)

scores = []
for i in range(num_features):
    score = test.scores_[i]
    scores.append((score, X_train2.columns[i]))

print (sorted(scores, reverse = True))
```

```
[(0.6671626913475868, 'health'), (0.15002428064610718, 'has_nurs'),
(0.049082432264722975, 'parents'), (0.019430268361006142, 'social'),
(0.01479428914029901, 'housing'), (0.008336981751714045, 'form'),
(0.008329247089529979, 'finance'), (0.0049258133024050466, 'childre
n')]
```

2. Chi-squared

In [20]:

```
from sklearn.feature_selection import chi2
test = SelectKBest(score_func = chi2, k=2)
```

Nursery dataset

In [21]:

```
test.fit(X_train2, y_train2)
num_features = len(X_train2.columns)

scores = []
for i in range(num_features):
    score = test.scores_[i]
    scores.append((score, X_train2.columns[i]))

print (sorted(scores, reverse = True))
```

```
[(2330.18746786419, 'health'), (1411.86832073306, 'has_nurs'), (277.
77034581620404, 'parents'), (77.60311192178953, 'housing'), (65.4291
6585994789, 'children'), (54.447857258526255, 'social'), (31.3154289
26512134, 'form'), (8.674972717871512, 'finance')]
```

Dimension reduction

PCA - Banknotes dataset

Check to see the overall weight of each principal component has on the variance of values.

In [22]:

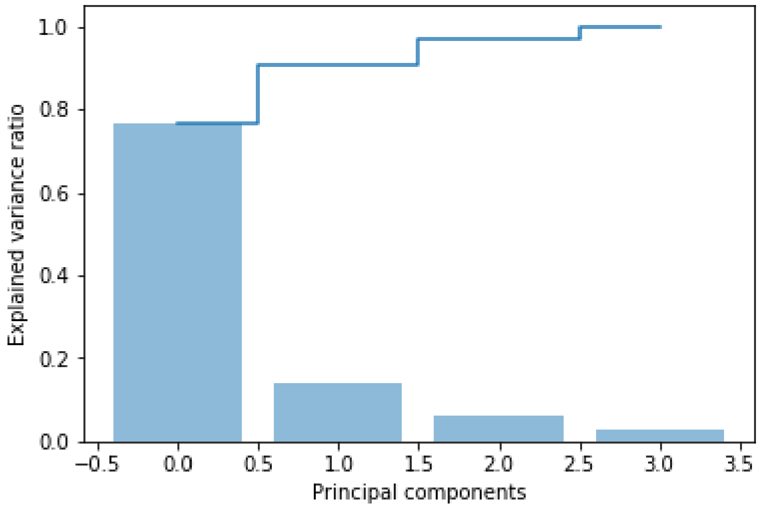
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

pca = PCA()
principalComponents = pca.fit_transform(X_train1)

print("Explained variance ratio:",pca.explained_variance_ratio_)
range_value = pca.explained_variance_ratio_.shape[0]

plt.bar(range(range_value), pca.explained_variance_ratio_, alpha=0.5, align='cen
ter')
plt.step(range(range_value), np.cumsum(pca.explained_variance_ratio_), where='mi
d')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.show()
```

```
Explained variance ratio: [0.76650539 0.1412553  0.06298564 0.029253
68]
```



In [23]:

```
# The eigenvectors
print('eigenvectors\n', pca.components_)
# singular values
print('singular_ values\n', pca.singular_values_)
```

```
eigenvectors
 [[-0.13990472 -0.81245605  0.54669634  0.14650921]
 [-0.78690389  0.26645552  0.31717771 -0.45736421]
 [ 0.45600508  0.44257498  0.77208222  0.00870563]
 [ 0.39149495 -0.27026657 -0.06641119 -0.87708451]]
singular_ values
 [222.008877    95.30487265  63.64048863  43.37135357]
```

Drawing graph using first 2 components.

In [24]:

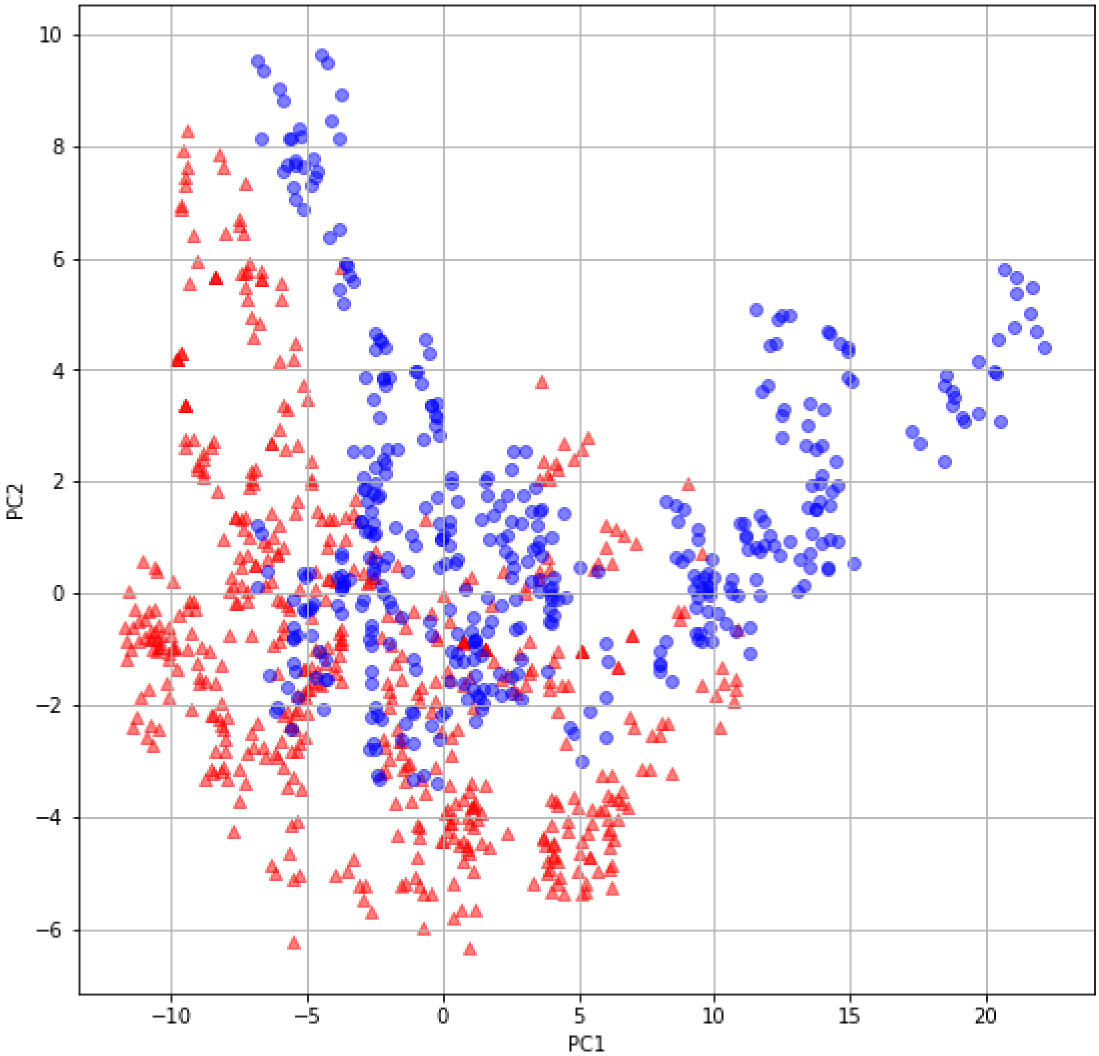
```
pca = PCA(n_components=2)
X_train_pca1 = pca.fit_transform(X_train1)

fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(2,2,1)

ax.scatter(X_train_pca1[y_train1 == 0, 0], X_train_pca1[y_train1 == 0, 1],
              color='red', marker='^', alpha=0.5)
ax.scatter(X_train_pca1[y_train1 == 1, 0], X_train_pca1[y_train1 == 1, 1],
              color='blue', marker='o', alpha=0.5)

ax.set_xlabel('PC1')
ax.set_ylabel('PC2')

ax.grid()
plt.show()
```



KPCA - Banknotes dataset

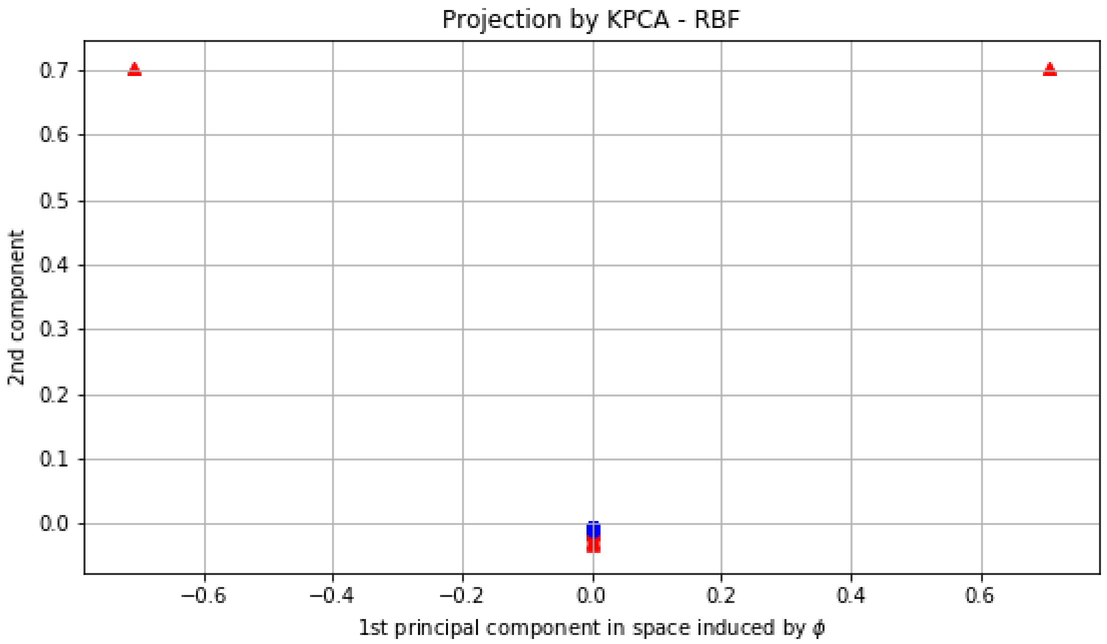
In [25]:

```
from sklearn.decomposition import KernelPCA

kpca = KernelPCA( n_components = 2, kernel="rbf", fit_inverse_transform=True, ga
mma=10)
X_train_kpca1 = kpca.fit_transform(X_train1)
fig = plt.figure(figsize=(20, 20))

plt.subplot(2, 2, 1, aspect='equal')

plt.scatter(X_train_kpca1[y_train1 == 0, 0], X_train_kpca1[y_train1 == 0, 1], c=
"red",
            marker='^', alpha=0.5)
plt.scatter(X_train_kpca1[y_train1 == 1, 0], X_train_kpca1[y_train1 == 1, 1], c=
"blue",
            marker='o', alpha=0.5)
plt.title("Projection by KPCA - RBF")
plt.xlabel("1st principal component in space induced by $\phi$")
plt.ylabel("2nd component")
plt.grid()
plt.show()
```



PCA - Nursery dataset

Unlike with the first dataset, the principal components here are much similar.

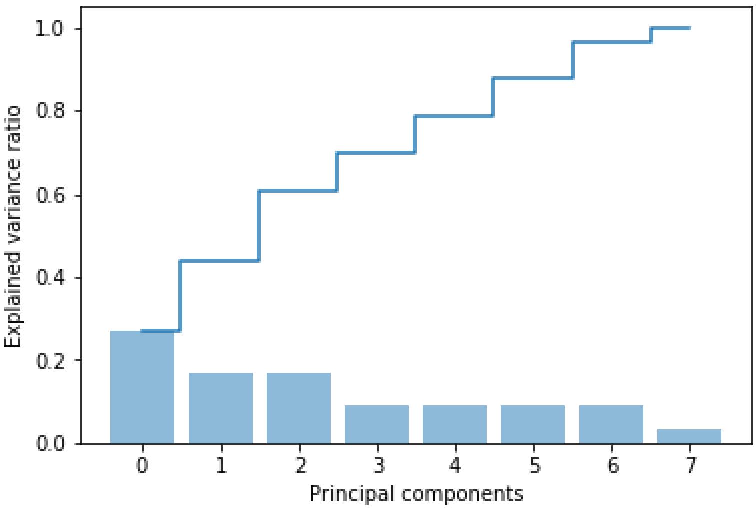
In [26]:

```
pca = PCA()
principalComponents = pca.fit_transform(X_train2)

print("Explained variance ratio:",pca.explained_variance_ratio_)
range_value = pca.explained_variance_ratio_.shape[0]

plt.bar(range(range_value), pca.explained_variance_ratio_, alpha=0.5, align='cen
ter')
plt.step(range(range_value), np.cumsum(pca.explained_variance_ratio_), where='mi
d')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.show()
```

```
Explained variance ratio: [0.26999765 0.1687268  0.16783652 0.091566
63 0.09001125 0.08938557
 0.08874903 0.03372655]
```



In [27]:

```
# The eigenvectors
print('eigenvectors\n', pca.components_)
# singular values
print('singular_ values\n', pca.singular_values_)
```

```
eigenvectors
 [[-9.47934855e-04  9.99987031e-01 -4.69267657e-04  3.15527592e-03
   2.19444550e-03 -1.12106606e-03  1.73251965e-03 -2.40617464e-03]
 [ 1.89466735e-02 -2.46758684e-03  4.68891697e-01  8.82783530e-01
  -1.41258805e-02  2.15045191e-04 -9.84467705e-03  1.31242379e-02]
 [-3.78226021e-03  1.87240217e-03  8.83135924e-01 -4.68854330e-01
   1.34920347e-02 -2.89003610e-04 -6.82737577e-03  1.13992559e-04]
 [ 4.12171541e-01 -2.90528484e-03  7.30183278e-04  1.25106793e-02
   6.20405745e-01  2.30800003e-03  2.21596985e-01 -6.29233892e-01]
 [-5.72602922e-01 -2.17018459e-03  1.06162531e-02  1.71697453e-02
   1.48186704e-01  7.86780525e-03  8.04181445e-01  5.46322414e-02]
 [ 1.18211569e-01  4.83622346e-04 -9.70306032e-03  1.85070171e-03
   6.80872099e-01 -4.55751871e-03 -8.98764082e-02  7.17107287e-01]
 [-6.98438277e-01 -1.27327421e-03 -1.97653752e-03  2.00943796e-02
   3.59389314e-01  6.65751874e-03 -5.44012446e-01 -2.94312958e-01]
 [-8.73680764e-03 -1.15666158e-03 -3.75980808e-05  6.11149495e-04
   1.87808287e-03 -9.99933143e-01  3.62458534e-03 -6.24499060e-03]]
singular_ values
 [134.72987866 106.50646044 106.22510316  78.46076332  77.79153211
  77.52068959  77.24417378  47.61786029]
```

Drawing a graph with the first 2 PCs doesn't show us much variance. There are a lot of points stacked on top of each other.

In [28]:

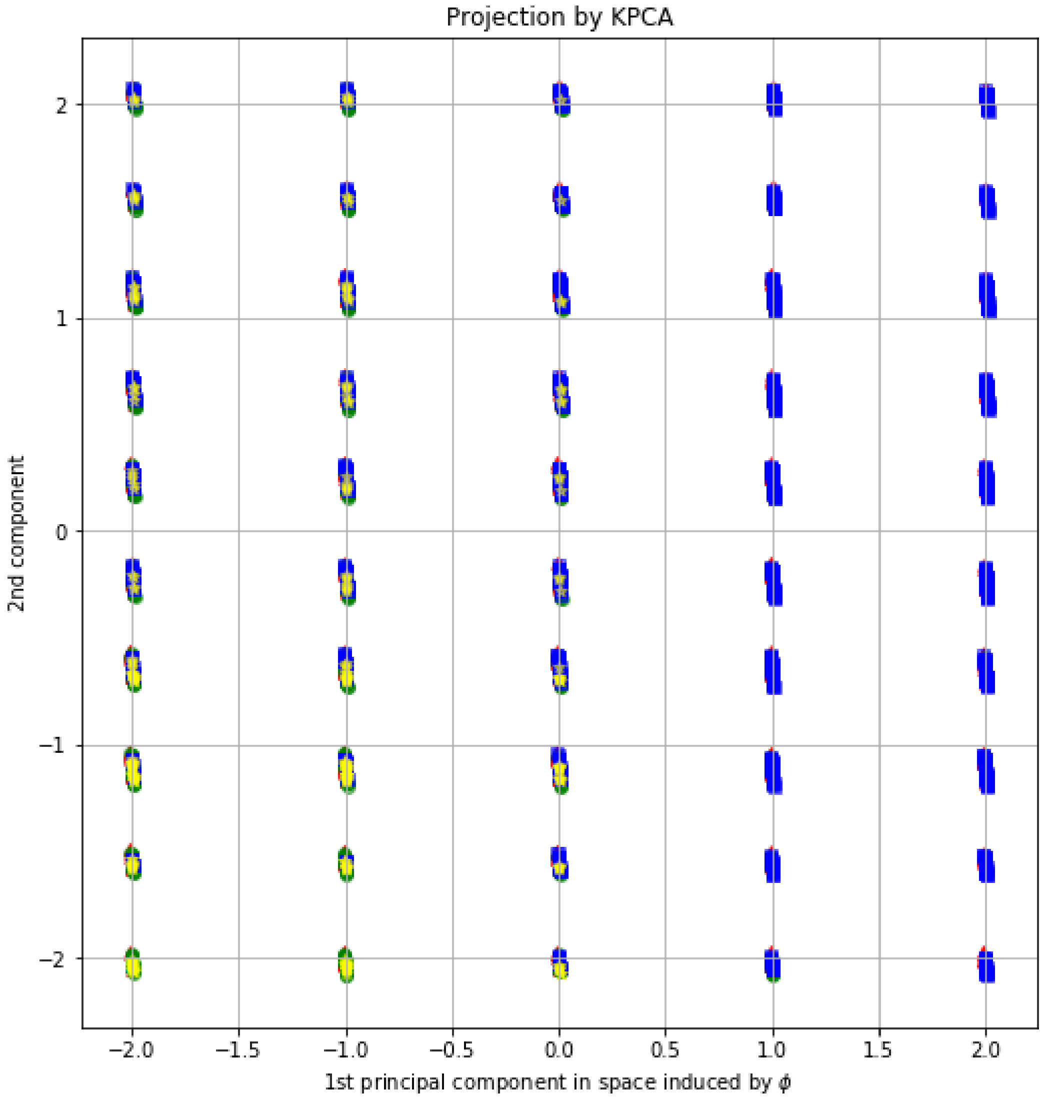
```
# Only two components for illustration
pca = PCA(n_components=7)
X_train_pca2 = pca.fit_transform(X_train2)

X_train2_v1 = pd.DataFrame(data=X_train_pca2)

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

plt.subplot(2, 2, 1, aspect='equal')

plt.scatter(X_train_pca2[y_train2 == 0, 0], X_train_pca2[y_train2 == 0, 1], c="red",
            marker='^', alpha=0.5)
plt.scatter(X_train_pca2[y_train2 == 1, 0], X_train_pca2[y_train2 == 1, 1], c="green",
            marker='o', alpha=0.5)
plt.scatter(X_train_pca2[y_train2 == 2, 0], X_train_pca2[y_train2 == 2, 1], c="cyan",
            marker='x', alpha=0.5)
plt.scatter(X_train_pca2[y_train2 == 3, 0], X_train_pca2[y_train2 == 3, 1], c="blue",
            marker='s', alpha=0.5)
plt.scatter(X_train_pca2[y_train2 == 4, 0], X_train_pca2[y_train2 == 4, 1], c="yellow",
            marker='*', alpha=0.5)
plt.title("Projection by KPCA")
plt.xlabel("1st principal component in space induced by $\phi$")
plt.ylabel("2nd component")
plt.grid()
plt.show()
```



KPCA - Nursery dataset

In [29]:

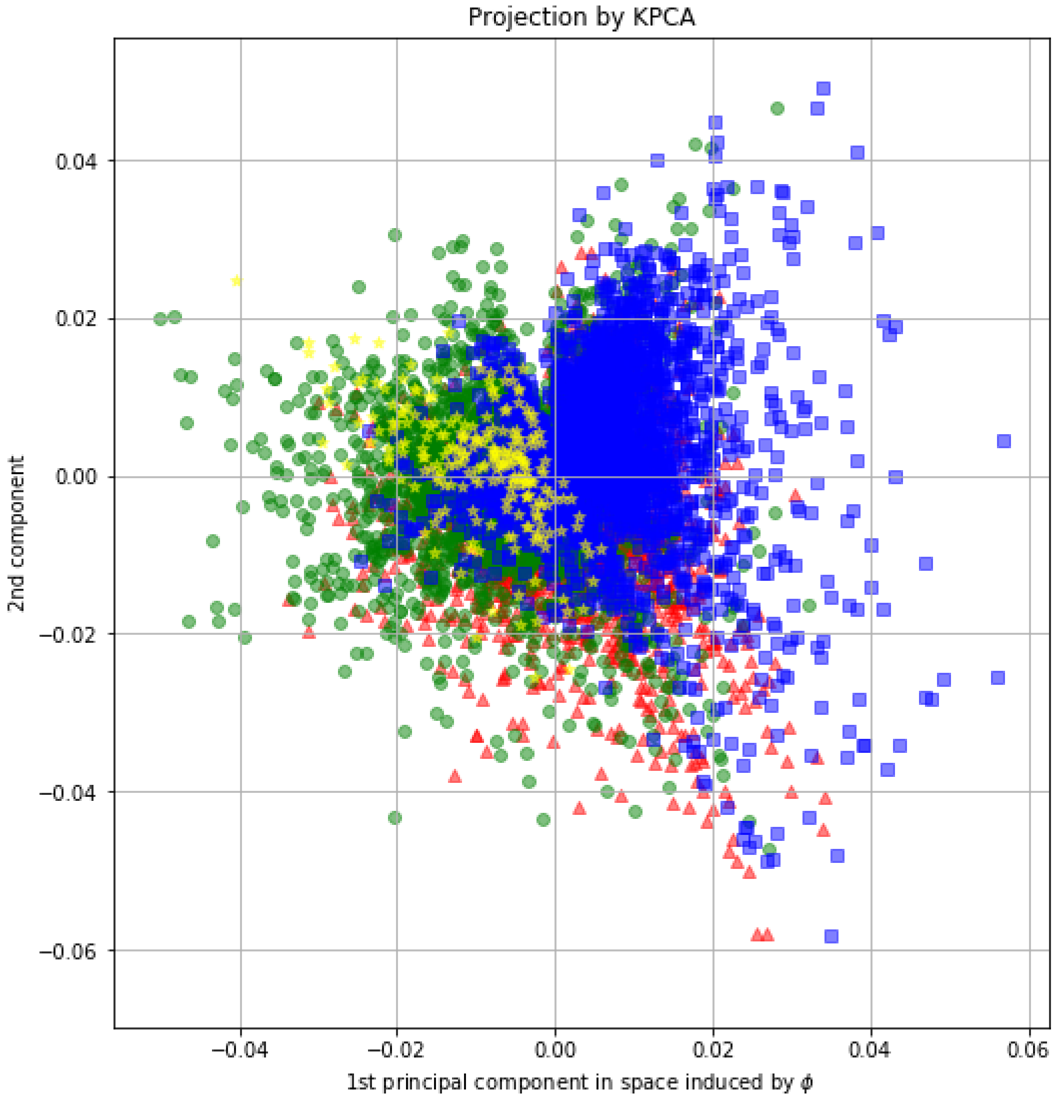
```
from sklearn.decomposition import KernelPCA

kpca = KernelPCA(n_components = 2 ,kernel="rbf", fit_inverse_transform=True, gam
ma=10)
X_train_kpca2 = kpca.fit_transform(X_train2)

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

plt.subplot(2, 2, 1, aspect='equal')

plt.scatter(X_train_kpca2[y_train2 == 0, 0], X_train_kpca2[y_train2 == 0, 1], c=
"red",
            marker='^', alpha=0.5)
plt.scatter(X_train_kpca2[y_train2 == 1, 0], X_train_kpca2[y_train2 == 1, 1], c=
"green",
            marker='o', alpha=0.5)
plt.scatter(X_train_kpca2[y_train2 == 2, 0], X_train_kpca2[y_train2 == 2, 1], c=
"cyan",
            marker='x', alpha=0.5)
plt.scatter(X_train_kpca2[y_train2 == 3, 0], X_train_kpca2[y_train2 == 3, 1], c=
"blue",
            marker='s', alpha=0.5)
plt.scatter(X_train_kpca2[y_train2 == 4, 0], X_train_kpca2[y_train2 == 4, 1], c=
"yellow",
            marker='*', alpha=0.5)
plt.title("Projection by KPCA")
plt.xlabel("1st principal component in space induced by $\phi$")
plt.ylabel("2nd component")
plt.grid()
plt.show()
```



Task B - Predictive Model

The models chosen were MLP and SVM.

In [30]:

```
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, 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=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
            alpha=0.8, c=cmap(idx), marker=markers[idx], label=cl)
```

MLP - Banknotes dataset

In [31]:

```
from sklearn.neural_network import MLPClassifier

mlp_Dataset1 = MLPClassifier(activation='tanh', hidden_layer_sizes=(10,5), alpha
=0.01, max_iter=5000)
mlp_Dataset1
```

Out[31]:

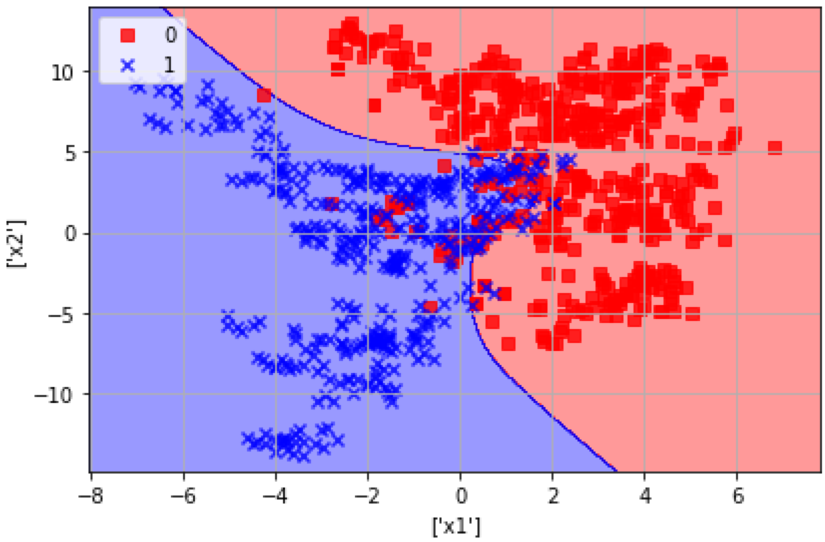
```
MLPClassifier(activation='tanh', alpha=0.01, batch_size='auto', beta
_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(10, 5), learning_rate='constant',
       learning_rate_init=0.001, max_iter=5000, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
```

In [32]:

```
X12=X_train1[X_train1.columns[0:2]]

mlp_Dataset1 = mlp_Dataset1.fit(X12.values,y_train1.values)
plot_decision_regions(X12.values, y_train1.values, classifier=mlp_Dataset1)
plt.xlabel(['x1'])
plt.ylabel(['x2'])
plt.legend(loc='upper left')
plt.grid()
plt.tight_layout()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
```



In [33]:

```
print('Banknotes dataset')
print('the weights are \n',mlp_Dataset1.coefs_)
print('the bias \n ', mlp_Dataset1.intercepts_)
print('number of iterations \n', mlp_Dataset1.n_iter_)
print('output activation', mlp_Dataset1.out_activation_)
```

```
Banknotes dataset
the weights are
 [array([[ 0.27912839, -0.44955468,  0.62629801, -0.17840014,  0.474
76466,
        -0.21131811, -0.39922233,  0.38770192, -0.18936666, -0.14676
214],
       [ 0.10251168, -0.69887619, -0.22839388, -0.08354042,  0.17291
558,
        -0.0077546 , -0.15510242,  0.2198039 ,  0.28318979, -0.08332
09 ]]), array([[ 0.50021791,  0.28354575,  0.52524301, -0.09777124,
0.3895508 ],
       [ 1.04568631, -0.40775077,  0.32993053,  0.56264259,  1.11460
67 ],
       [ 0.57165465, -0.77688242,  1.00035204,  0.8872225 ,  0.13308
413],
       [-0.49522631,  0.84165537, -0.29681432, -0.75865147, -0.82262
961],
       [ 0.36526798, -0.75224419, -0.07284236,  0.39400805,  0.44518
479],
       [-0.217362  , -0.14477429,  0.19645376, -0.16894034,  0.11305
763],
       [-0.54110794, -0.11527716, -0.55744865, -0.76031779, -0.63174
027],
       [-0.38152757, -0.53225613,  0.20481981,  0.30668199,  0.32003
436],
       [ 0.81957294, -0.76918728,  0.90570855,  1.28662406,  0.37007
535],
       [-0.67234703, -0.12597384, -0.18677749, -0.3254005 , -0.24962
765]]), array([[-0.41034833],
       [ 1.10939539],
       [-1.58115178],
       [-1.23598788],
       [-1.32530678]])]
the bias
  [array([ 0.52039531,  0.81119956, -0.93971943, -0.02059897,  1.034
64378,
        0.70490408, -0.75028911, -1.48351233, -1.56348119, -0.214595
89]), array([ 0.15349875, -0.5185196 ,  0.10526876, -0.05685343,  0.
3064959 ]), array([-0.29217585])]
number of iterations
 405
output activation logistic
```

MLP - Nursery dataset

In [34]:

```
mlp_Dataset2 = MLPClassifier(activation='tanh', hidden_layer_sizes=(10,5), alpha
=0.01, max_iter=5000)
mlp_Dataset2
```

Out[34]:

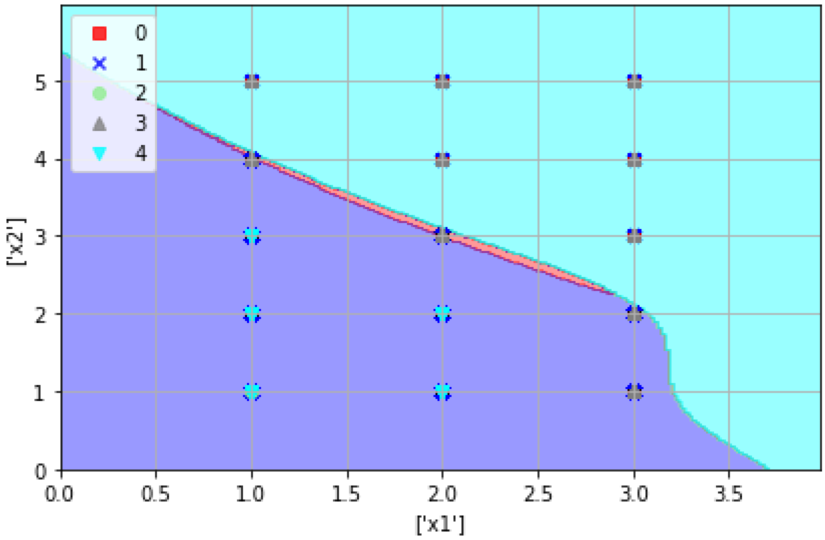
```
MLPClassifier(activation='tanh', alpha=0.01, batch_size='auto', beta
_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(10, 5), learning_rate='constant',
       learning_rate_init=0.001, max_iter=5000, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
```

In [35]:

```
X22=X_train2[X_train2.columns[0:2]]
mlp_Dataset2 = mlp_Dataset2.fit(X22.values,y_train2.values)

plot_decision_regions(X22.values, y_train2.values, classifier=mlp_Dataset2)
plt.xlabel(['x1'])
plt.ylabel(['x2'])
plt.legend(loc='upper left')
plt.grid()
plt.tight_layout()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
```



In [36]:

```
print('Nursery dataset')
print('the weights are \n',mlp_Dataset2.coefs_)
print('the bias \n ', mlp_Dataset2.intercepts_)
print('number of iterations \n', mlp_Dataset2.n_iter_)
print('output activation', mlp_Dataset2.out_activation_)
```

```
Nursery dataset
the weights are
 [array([[ 0.39918774, -0.58851606,  0.56942324,  0.81525291,  0.390
70391,
         0.11038901,  0.32118373,  0.52462088, -0.36008977,  0.11460
914],
       [ 0.40757144, -0.7484958 ,  0.2405804 , -1.33827025,  0.45045
278,
         0.39943511,  0.17918946,  0.53433873, -0.13137104, -0.47177
555]]), array([[ 0.22309919, -0.73749193,  0.45534885, -0.51019626,
0.13228479],
       [ 0.73130393,  0.02766277, -0.64951006,  0.73388908, -0.21453
857],
       [ 1.16428322,  0.22821304, -1.17696545, -0.26605887,  2.07651
283],
       [ 0.00321566,  0.58106586,  0.29848459, -0.45905575,  1.37510
809],
       [-0.37345229, -0.34376359, -0.35059715, -0.47162803, -0.31962
718],
       [ 1.01285972,  0.09246268, -0.72704582,  0.11831987,  1.69289
266],
       [ 0.57686872, -0.18012712, -1.10457792, -0.19109844,  0.92884
311],
       [ 0.17955787, -0.49136034,  0.6717721 , -0.34667568, -0.12483
113],
       [-0.39738367,  0.41683773,  0.44574193,  0.45189584, -0.78112
884],
       [-0.66355849, -0.15242789,  0.7836701 ,  0.0066445 , -0.84945
612]]), array([[-0.46931521,  0.32639376, -1.36987529,  0.41378822,
0.01103675],
       [-0.89234145, -1.14482369,  1.7397929 , -0.19911418,  0.58669
503],
       [-0.73602268,  0.90909283, -0.94874197, -0.09850948,  0.47109
101],
       [-1.11075903, -0.17734299,  1.90217783,  0.39632715,  1.04205
164],
       [-0.21413316, -0.50327988, -0.55610716,  2.51160764, -2.28244
924]])]
the bias
  [array([ 0.96394415, -0.21509335, -1.4179756 , -0.59643627,  0.875
61933,
       -1.23634074, -0.72636172,  0.5858667 ,  0.39116638,  1.134515
41]), array([ 0.04100978, -0.41790828,  0.55493849, -0.09153707,  0.
02825155]), array([ 0.99341884,  0.98367834, -0.53481707,  0.7032301
8,  0.36439277])]
number of iterations
 231
output activation softmax
```

SVM - Banknotes dataset

In [37]:

```
from sklearn.svm import SVC

svm_Dataset1=SVC(C=1.0,kernel='rbf', tol=1e-05, verbose=0)
#svm_Dataset1=SVC(C=1.0,kernel='rbf', max_iter=2000, tol=1e-05, verbose=0)

svm_Dataset1
```

Out[37]:

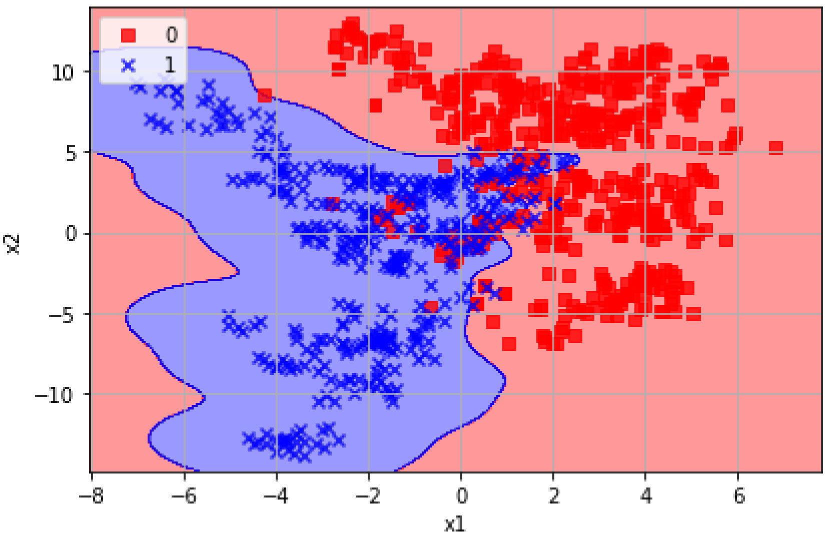
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=1e-05, verbose=0)
```

In [38]:

```
X12=X_train1[X_train1.columns[0:2]]

svm_Dataset1 = svm_Dataset1.fit(X12.values, y_train1.values)
plot_decision_regions(X12.values, y_train1.values, classifier=svm_Dataset1)
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(loc='upper left')
plt.grid()
plt.tight_layout()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
```



In [39]:

```
print('Banknotes dataset')
print('dual coef \n', svm_Dataset1.dual_coef_)
print ('support vectors \n', svm_Dataset1.support_vectors_)
print('index of support vectors \n ', svm_Dataset1.support_)
print ('bias', svm_Dataset1.intercept_)
print('the classifier \n', svm_Dataset1)
```

```
Banknotes dataset
dual coef
 [[-1.00000000e+00 -2.56542151e-01 -1.00000000e+00 -1.00000000e+00
  -2.44640581e-01 -2.93953106e-01 -7.78827102e-01 -1.00000000e+00
  -1.00000000e+00 -6.61899850e-04 -3.19717089e-01 -1.00000000e+00
  -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -5.55779036e-02
  -1.00000000e+00 -9.68983664e-02 -1.00000000e+00 -4.50753209e-02
  -1.17662606e-01 -7.54783428e-02 -1.00000000e+00 -1.00000000e+00
  -2.16126231e-02 -1.00000000e+00 -1.00000000e+00 -4.73509741e-01
  -4.87760530e-01 -8.23227722e-01 -1.00000000e+00 -5.75946278e-01
  -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
  -1.00000000e+00 -5.08635095e-01 -1.00000000e+00 -1.00000000e+00
  -1.00000000e+00 -6.37617903e-01 -3.43756379e-01 -1.02373314e-01
  -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
  -2.90681697e-03 -5.10955042e-01 -1.00000000e+00 -1.00000000e+00
  -3.31906131e-01 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
  -2.49867603e-03 -1.39247088e-02 -1.00000000e+00 -3.91669611e-01
  -4.76571100e-01 -7.46593726e-01 -1.00000000e+00 -5.66477528e-01
  -1.00000000e+00 -4.02967742e-01 -3.04351283e-02 -2.04789094e-01
  -1.00000000e+00 -1.10172656e-01 -1.00000000e+00 -1.90904672e-01
  -1.00000000e+00 -5.84620020e-02 -1.86055313e-01 -5.87629762e-01
  -1.00000000e+00 -1.00000000e+00 -8.47424484e-01 -5.94285662e-01
  -1.00000000e+00 -1.13960502e-01 -7.07435870e-01 -6.00143998e-01
  -1.91248651e-01 -1.00000000e+00 -1.00000000e+00 -3.18084700e-02
  -8.33367645e-01 -9.20772997e-02 -1.00000000e+00 -1.00000000e+00
  -1.00921167e-02 -1.00000000e+00 -1.00000000e+00 -3.40906589e-01
  -1.00000000e+00 -1.88615980e-02 -1.00000000e+00 -8.77299309e-02
  -2.12656347e-02 -1.00000000e+00 -2.63558346e-01 -4.99187091e-01
  -7.06177704e-02 -1.00000000e+00 -1.00000000e+00 -5.52515895e-02
  -1.00000000e+00 -1.31887965e-01 -1.00000000e+00 -1.00000000e+00
  -8.43155226e-01 -1.00000000e+00 -1.11715896e-01 -3.21647233e-01
  -1.00000000e+00 -3.79708979e-01 -4.55604225e-02 -1.59468288e-01
  -3.92675949e-01 -5.96319196e-02 -1.00000000e+00 -1.00000000e+00
  -3.17362666e-01 -5.56843808e-01 -1.00000000e+00 -1.00000000e+00
  -1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -1.00000000e+00
  -9.88645317e-01 -1.98378473e-01 -1.00000000e+00 -1.98521584e-01
  -1.00000000e+00 -1.86928802e-01 -1.00000000e+00 -2.15281959e-01
  -1.00000000e+00 -3.35559758e-01 -1.00000000e+00 -1.00000000e+00
  -6.70795923e-01  4.45693159e-01  1.00000000e+00  1.00000000e+00
   1.00000000e+00  1.00000000e+00  6.74667355e-02  5.02483811e-01
   8.00856531e-01  8.08429084e-01  7.11836909e-01  5.88878469e-01
   1.88701290e-01  3.40638705e-01  1.00000000e+00  5.31559782e-01
   1.00000000e+00  1.00000000e+00  8.25263854e-01  1.00000000e+00
   1.00000000e+00  1.00000000e+00  1.00000000e+00  4.16242513e-01
   1.00000000e+00  1.35216996e-01  1.00000000e+00  1.00000000e+00
   8.57845848e-01  1.42831646e-01  1.00000000e+00  1.00000000e+00
   1.00000000e+00  2.28301250e-01  5.37844774e-02  8.22833126e-01
   1.00000000e+00  3.92709069e-01  3.98851923e-01  1.00000000e+00
   1.00000000e+00  7.50006232e-01  1.00000000e+00  1.00000000e+00
   1.00000000e+00  1.00000000e+00  1.00000000e+00  8.64608637e-01
   3.37611022e-01  1.00000000e+00  1.00000000e+00  1.00000000e+00
   1.57972898e-01  2.86599583e-01  1.00000000e+00  3.86583595e-01
   1.00000000e+00  2.76140534e-01  2.08484620e-01  2.47667894e-01
   6.09881888e-01  3.00343312e-02  1.00000000e+00  1.00000000e+00
   1.00000000e+00  6.04924793e-01  6.68457836e-01  1.00000000e+00
   1.00000000e+00  5.88787818e-01  1.14219069e-01  1.00000000e+00
   1.00000000e+00  1.00000000e+00  9.83237684e-01  1.00000000e+00
   1.35324000e-01  8.39126609e-01  4.51277262e-01  1.00000000e+00
   1.00000000e+00  1.00000000e+00  6.21676004e-02  1.00000000e+00
   6.51448939e-01  1.00000000e+00  9.02609395e-01  1.00000000e+00
   1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
```

```
   1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
   1.34677493e-01  1.00000000e+00  1.00000000e+00  3.75252650e-01
   6.47353258e-01  1.00000000e+00  6.01469123e-01  4.23671048e-02
   3.16302092e-01  1.00000000e+00  2.52853492e-01  3.31364355e-01
   1.00000000e+00  2.50797981e-01  1.76863586e-01  1.00000000e+00
   1.00000000e+00  1.00000000e+00  3.16498852e-01  1.00000000e+00
   1.00000000e+00  6.41622189e-01  1.00000000e+00  1.00000000e+00
   2.17050289e-01  4.18048400e-01  1.00000000e+00  7.34726082e-01
   7.63862844e-01  9.43196828e-01  1.00000000e+00  9.85483145e-01]]
support vectors
 [[ 5.70600e-01 -2.48000e-02]
 [ 1.92650e+00  7.75570e+00]
 [ 5.19500e-01 -3.26330e+00]
 [ 2.25040e+00  3.57570e+00]
 [ 4.92640e+00  5.49600e+00]
 [-1.38850e+00  1.25026e+01]
 [ 4.96650e-01  5.52700e+00]
 [ 1.14720e+00  3.59850e+00]
 [ 1.74520e+00  4.80280e+00]
 [-2.64790e+00  1.01374e+01]
 [ 2.22900e+00  9.63250e+00]
 [ 7.40540e-01  3.66250e-01]
 [ 6.54970e-01  5.18150e+00]
 [ 5.28550e-01  9.64270e-01]
 [ 1.01910e+00  2.33000e+00]
 [ 4.05200e+00 -1.65550e-01]
 [-1.50550e+00  7.03460e-02]
 [ 2.95710e+00 -4.59380e+00]
 [ 2.00070e+00  1.86440e+00]
 [-1.32740e+00  9.49800e+00]
 [ 9.29700e-01 -3.79710e+00]
 [ 2.56780e+00  3.51360e+00]
 [ 3.29200e-01 -4.45520e+00]
 [ 1.31140e+00  4.54620e+00]
 [ 5.30630e+00  5.26840e+00]
 [-2.79140e+00  1.77340e+00]
 [-1.61620e+00  8.09080e-01]
 [ 5.40210e+00  3.10390e+00]
 [ 3.96600e+00  3.92130e+00]
 [ 1.04000e+00 -6.93210e+00]
 [ 1.77480e+00 -7.69780e-01]
 [ 5.59100e+00  1.04643e+01]
 [ 1.30870e+00  4.92280e+00]
 [ 1.59400e+00  4.70550e+00]
 [ 1.32640e+00  1.03260e+00]
 [-1.77810e+00  8.54600e-01]
 [ 5.19500e-01 -3.26330e+00]
 [ 5.49440e+00  1.54780e+00]
 [-1.31440e-01 -1.77750e+00]
 [ 2.04210e+00  1.24360e+00]
 [ 1.85920e+00  3.20740e+00]
 [ 5.78670e+00  7.89020e+00]
 [ 2.44860e+00 -6.31750e+00]
 [ 5.17310e+00  3.96060e+00]
 [-4.50620e-01 -1.36780e+00]
 [ 1.14300e+00  8.33910e-01]
 [ 1.35660e+00  4.23580e+00]
 [ 1.63490e+00  3.28600e+00]
 [-2.64790e+00  1.01374e+01]
 [-2.64790e+00  1.01374e+01]
 [ 5.70600e-01 -2.48000e-02]
```

```
 [ 7.05700e-01 -5.49810e+00]
 [ 4.15420e+00  7.27560e+00]
 [ 1.26160e+00  4.43030e+00]
 [ 1.32340e+00  3.29640e+00]
 [-1.13130e+00  1.90370e+00]
 [-1.85840e+00  7.88600e+00]
 [-5.39660e-01  7.32730e+00]
 [ 5.70600e-01 -2.48000e-02]
 [ 4.16650e+00 -4.44900e-01]
 [ 4.25860e+00  1.12962e+01]
 [-1.85840e+00  7.88600e+00]
 [ 7.57360e-01  3.02940e+00]
 [-3.98160e-01  5.97810e+00]
 [ 5.19500e-01 -3.26330e+00]
 [ 4.89060e+00 -3.35840e+00]
 [ 8.72560e-01  9.29310e+00]
 [ 3.54580e+00  9.37180e+00]
 [-4.28590e+00  8.52340e+00]
 [-9.17180e-01  9.98840e+00]
 [-1.33890e+00  1.55200e+00]
 [ 9.27030e-01  9.43180e+00]
 [-1.69520e+00  1.06570e+00]
 [ 2.25460e+00  8.09920e+00]
 [ 3.81970e+00  8.99510e+00]
 [ 7.43070e-01  1.11700e+01]
 [ 5.70600e-01 -2.48410e-02]
 [ 1.59020e+00  2.29480e+00]
 [ 2.67180e+00  5.65740e+00]
 [ 5.02970e+00 -4.97040e+00]
 [-1.39310e+00  1.56640e+00]
 [ 3.89050e+00 -2.15210e+00]
 [ 6.82480e+00  5.21870e+00]
 [-2.34300e+00  1.29516e+01]
 [ 3.92940e+00  1.41120e+00]
 [ 1.89940e+00  9.74620e-01]
 [ 1.09870e+00  6.39400e-01]
 [ 4.28990e+00  9.18140e+00]
 [ 1.64720e+00  4.82130e-01]
 [ 2.74510e-01  9.21860e+00]
 [-1.06480e-01 -7.67710e-01]
 [ 3.79800e-01  7.09800e-01]
 [ 1.00090e+00  7.78460e+00]
 [ 1.76200e+00  4.36820e+00]
 [-3.83880e-01 -1.04710e+00]
 [ 3.03330e+00 -2.59280e+00]
 [ 3.29240e-01 -4.45520e+00]
 [ 1.86640e+00  7.77630e+00]
 [-1.17830e-01 -1.57890e+00]
 [ 3.13770e+00 -4.10960e+00]
 [-6.89190e-03  9.29310e+00]
 [-9.59230e-01  9.10390e-02]
 [ 5.93740e+00  6.16640e+00]
 [ 3.40920e+00  5.40490e+00]
 [-7.86900e-01  9.56630e+00]
 [ 1.29990e+00  2.57620e+00]
 [ 1.13170e+00  3.96470e+00]
 [ 1.54780e+00  9.18140e+00]
 [ 4.41250e-01  2.94870e+00]
 [ 1.48060e+00  7.63770e+00]
 [-2.48110e-01 -1.77970e-01]
 [-1.13910e+00  1.81270e+00]
```

```
 [ 2.01530e+00  1.84790e+00]
 [ 3.22300e-01 -8.98080e-01]
 [ 1.89670e+00 -2.51630e+00]
 [ 1.47830e-01  7.94600e+00]
 [ 3.79840e-01  7.09750e-01]
 [-2.74190e+00  1.14038e+01]
 [ 4.17110e+00  8.72200e+00]
 [ 4.24750e+00  1.48160e+00]
 [ 3.75700e+00 -5.42360e+00]
 [ 1.73460e-01  7.86950e+00]
 [ 2.28930e+00  3.73300e+00]
 [ 3.31110e-01  4.57310e+00]
 [ 4.55970e+00 -2.42110e+00]
 [ 2.80840e+00  1.13045e+01]
 [ 3.79800e-01  7.09800e-01]
 [ 3.79800e-01  7.09800e-01]
 [-3.60380e-01  4.11580e+00]
 [ 4.69010e-01 -6.33210e-01]
 [ 2.09770e-01 -4.61460e-01]
 [ 8.82980e-01  6.60090e-01]
 [ 9.29700e-01 -3.79710e+00]
 [ 3.23030e+00  7.83840e+00]
 [ 6.42150e-01  3.12870e+00]
 [-7.86900e-01  9.56630e+00]
 [ 7.22520e-01 -5.38110e-02]
 [ 2.87700e+00 -4.05990e+00]
 [-6.44720e-01 -4.60620e+00]
 [ 5.43800e+00  9.46690e+00]
 [ 1.64260e+00  3.01490e+00]
 [-7.82890e-01  1.13603e+01]
 [ 1.06070e+00  2.45420e+00]
 [-1.50750e+00  1.92240e+00]
 [ 5.74030e+00 -4.42840e-01]
 [-4.19580e+00 -8.18190e+00]
 [ 1.78750e+00  4.78000e+00]
 [-1.61760e+00  1.09260e+00]
 [-2.16680e+00  1.59330e+00]
 [ 1.73310e+00  3.95440e+00]
 [-3.01930e+00  1.77750e+00]
 [-1.39460e+00  2.31340e+00]
 [-4.63380e+00 -1.27509e+01]
 [-3.72440e+00  1.90370e+00]
 [ 4.05450e-03  6.29050e-01]
 [-6.39790e+00  6.44790e+00]
 [-2.19790e+00 -2.12520e+00]
 [-7.20680e-01 -6.75830e+00]
 [-1.84480e+00  1.25400e+00]
 [-3.59330e+00  2.29680e-01]
 [ 3.18030e-01 -9.93260e-01]
 [ 1.21980e+00  2.09820e+00]
 [-2.64060e+00 -4.41590e+00]
 [ 2.01770e+00  1.79820e+00]
 [-3.60250e-01 -4.44900e+00]
 [ 1.34510e+00  2.35890e-01]
 [ 5.08130e-01  4.77990e-01]
 [-6.32980e-01 -5.12770e+00]
 [-1.28520e-03  1.38630e-01]
 [-2.57010e+00 -6.84520e+00]
 [ 1.50770e+00  1.95960e+00]
 [ 3.76370e-01 -8.23580e-01]
 [-5.53550e-01 -7.92330e+00]
```

```
 [-1.84390e+00 -8.64750e+00]
 [ 1.06370e+00  3.69570e+00]
 [-1.57320e+00  1.06360e+00]
 [ 3.12010e-03 -4.00610e+00]
 [-2.83910e+00 -6.63000e+00]
 [ 4.73680e-01  3.36050e+00]
 [-5.06760e+00 -5.18770e+00]
 [-8.73400e-01  1.65330e+00]
 [-4.38760e+00 -7.72670e+00]
 [-2.65900e+00 -1.60580e+00]
 [ 1.05520e+00  1.18570e+00]
 [ 2.68770e-01  4.98700e+00]
 [-7.04210e+00  9.20000e+00]
 [ 2.03100e+00  1.85200e+00]
 [ 1.64080e+00  4.25030e+00]
 [-1.13060e+00  1.84580e+00]
 [ 2.22790e+00  4.09510e+00]
 [ 3.43400e-01  1.24150e-01]
 [-5.49010e+00  9.10480e+00]
 [-1.30660e+00  2.52440e-01]
 [ 9.82960e-01  3.42260e+00]
 [ 1.16400e+00  3.91300e+00]
 [ 6.00500e-01  9.99450e-01]
 [-3.46050e+00  2.69010e+00]
 [-4.73310e+00 -6.17890e+00]
 [ 5.62320e-01  1.00150e+00]
 [-4.39670e+00  4.96010e+00]
 [-1.27920e+00  2.13760e+00]
 [-2.89570e+00 -1.20205e+01]
 [-2.48350e+00 -7.44940e+00]
 [-2.82670e+00 -9.04070e+00]
 [-3.37930e+00 -1.37731e+01]
 [-1.08020e+00  2.19960e+00]
 [ 5.59390e-01 -3.10400e-01]
 [ 4.28300e-01 -9.49810e-01]
 [ 1.43290e-01 -1.08850e+00]
 [-3.09860e+00 -1.04602e+01]
 [-1.08330e+00 -3.12470e-01]
 [ 6.63650e-01 -4.55330e-02]
 [ 8.95120e-01  4.77380e+00]
 [-5.16610e+00  8.04330e+00]
 [-3.11580e+00 -8.62890e+00]
 [ 1.63580e-01 -3.35840e+00]
 [-3.70130e-01 -5.55400e+00]
 [-1.93890e+00  1.57060e+00]
 [ 6.25250e-02  2.93010e+00]
 [-4.14790e+00  7.12250e+00]
 [-3.60530e+00 -5.97400e+00]
 [-2.53140e-02 -1.73830e-01]
 [ 7.40670e-01  1.72990e+00]
 [ 2.19430e+00  4.55030e+00]
 [ 1.23090e+00  3.89230e+00]
 [ 1.58100e+00  8.69090e-01]
 [-4.47790e+00  7.37080e+00]
 [ 7.58960e-01  2.91760e-01]
 [-1.05550e+00  7.94590e-01]
 [ 1.59040e+00  2.21210e+00]
 [ 1.20800e+00  4.07440e+00]
 [-2.40370e-01 -1.78370e+00]
 [-1.36600e+00  1.84160e-01]
 [ 7.51080e-01  1.91610e+00]
```

```
 [ 3.00810e-01  1.73810e-01]
 [ 2.34600e-01 -4.51520e+00]
 [ 7.44280e-01 -3.77230e+00]
 [ 9.13150e-01  3.33770e+00]
 [ 1.16440e+00  3.80950e+00]
 [ 1.48960e+00  3.42880e+00]
 [-1.75490e+00 -8.07110e-02]
 [ 2.14310e-01 -6.95290e-01]
 [ 2.39170e+00  4.55650e+00]
 [-6.73870e+00  6.98790e+00]
 [-2.02850e+00  3.84680e+00]
 [-2.18880e-01 -2.20380e+00]
 [-1.84830e+00  3.10380e-01]
 [-3.99340e+00  5.83330e+00]
 [-6.60080e-01 -3.22600e+00]
 [ 1.55140e+00  3.80130e+00]
 [-2.98210e+00  4.19860e+00]
 [-3.81670e+00  5.14010e+00]
 [ 1.01170e+00  9.02200e-01]
 [-1.11880e+00  3.33570e+00]
 [-2.56500e+00 -5.78990e+00]
 [ 8.36250e-01  1.10710e+00]
 [ 8.89920e-01  2.26380e+00]
 [ 2.95200e-01  4.88560e+00]
 [-3.34580e+00 -5.04910e-01]
 [ 1.54230e-01  1.17940e-01]
 [ 3.90120e-01 -1.42790e-01]
 [-1.48000e+00 -1.05244e+01]
 [ 1.43780e+00  6.68370e-01]
 [ 1.25720e+00  4.87310e+00]
 [-1.39680e+00 -9.66980e+00]
 [-2.66490e+00 -1.28130e+01]
 [ 9.04070e-01  3.37080e+00]
 [-7.15030e-02  3.74120e+00]
 [-4.94470e+00  3.30050e+00]
 [-1.26900e-01 -1.15050e+00]
 [ 5.52980e-01 -3.46190e+00]
 [-1.16670e+00 -1.42370e+00]]
index of support vectors
  [  9  18  38  46  57  63  67  68  73  76  77  85  86  90  92  97 1
00 104
 108 110 122 125 136 141 143 146 147 156 159 160 163 168 177 178 179
182
 183 190 196 217 219 239 246 260 269 279 293 316 323 325 327 335 336
338
 341 350 351 356 375 386 389 392 399 405 420 423 426 456 459 473 475
508
 542 552 553 556 561 574 578 583 590 595 596 607 611 619 620 624 635
639
 653 660 662 666 674 677 680 681 683 686 687 688 692 695 708 709 717
720
 721 733 740 742 752 757 761 767 774 778 782 805 810 820 821 822 826
831
 840 847 852 857 871 874 900 908 913 915 916 922 923 928 929 942 953
955
 959   6   7  11  12  16  17  28  29  32  41  71  75  78 106 107 118
120
 149 162 173 180 185 204 215 226 233 237 249 250 254 258 259 262 268
273
 290 294 300 303 309 311 312 313 317 320 332 333 337 340 346 354 361
380
```

```
 381 382 390 402 408 418 432 433 434 442 444 448 460 470 471 476 477
481
 490 494 497 510 513 539 551 555 562 566 571 579 584 587 598 601 616
618
 640 647 650 658 671 679 689 711 718 727 741 763 770 783 798 800 811
833
 834 836 839 845 848 854 855 862 864 875 890 893 895 899 905 906 912
939
 940 941]
bias [-0.08624007]
the classifier
 SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=1e-05, verbose=0)
```

SVM - Nursery dataset

In [40]:

```
svm_Dataset2=SVC(C=1.0,kernel='rbf', tol=1e-05, verbose=0)
#svm_Dataset2=SVC(C=1.0,kernel='rbf', max_iter=2000, tol=1e-05, verbose=0)
svm_Dataset2
```

Out[40]:

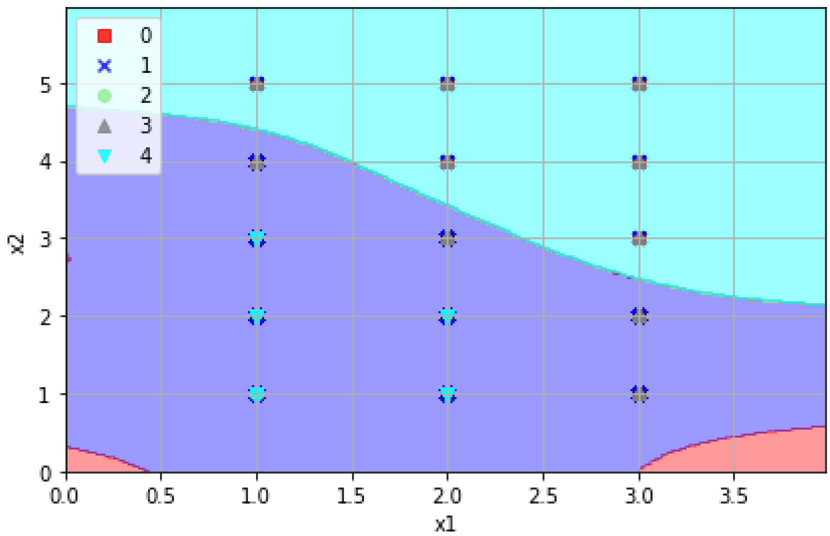
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=1e-05, verbose=0)
```

In [41]:

```
X22=X_train2[X_train2.columns[0:2]]

svm_Dataset2 = svm_Dataset2.fit(X22.values, y_train2.values)
plot_decision_regions(X22.values, y_train2.values, classifier=svm_Dataset2)
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(loc='upper left')
plt.grid()
plt.tight_layout()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single
row if you really want to specify the same RGB or RGBA value for all
points.
```



In [42]:

```
print('Nursery dataset')
print('dual coef \n', svm_Dataset2.dual_coef_)
print ('support vectors \n', svm_Dataset2.support_vectors_)
print('index of support vectors \n ', svm_Dataset2.support_)
print ('bias', svm_Dataset2.intercept_)
print('the classifier \n', svm_Dataset2)
```

```
Nursery dataset
dual coef
 [[ 0.          0.          1.         ... -1.         -1.
  -1.        ]
 [ 0.          0.          0.         ... -1.         -1.
  -1.        ]
 [ 1.          1.          0.         ... -0.         -0.
  -0.        ]
 [ 0.          0.          0.         ... -0.47985772 -0.
  -1.        ]]
support vectors
 [[3. 5.]
 [3. 4.]
 [2. 2.]
 ...
 [2. 2.]
 [1. 2.]
 [2. 1.]]
index of support vectors
  [   0    4    6 ... 8924 9023 9030]
bias [ 0.06711158  1.          0.21637831  1.00000039  1.
0.21637859
  1.00000039 -0.59885101 -1.          0.48176958]
the classifier
 SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=1e-05, verbose=0)
```