Departamento de Eletrónica, Telecomunicações e Informática

Mestrado Integrado em Engenharia de Computadores e Telemática

Data Mining assignment of Exploração de Dados (2018/2019)

Authors: Inês Lemos (76769) and Cristiana Carvalho (77682)

# Credit Card Dataset

The first dataset used was the one retrieved from the following link:
https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients
(https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients)

## Task A ¶

On Task A, the project stated that we had to follow the steps:

1. Mapping categorial data into numeric data
2. Rank features
3. Perform dimension reduction (Principal Component Analysis or Kernel Principal Components)

In this dataset, there was no categorical data, which means that the first step was suppressed.

The data was loaded on the **Build table** section. Then, on **Rank features (Univariate Feature Selection)**, we chose a univariate feature selection to find the scores (rank) of the best features, chosen by SelectKBest function.

Finally, we had to perform a dimension reduction on our data. For that, we used the well-known procedure **PCA**. After choosing X as all values of the table and y as the column 23 (the output column), we split X and y into random train and test subsets, standardized the data and only then we applied the PCA to the sets. The scatter plot was computed afterwards, showing the predicted values and with the 2 principal components in the axes.

The output column (Y on the excel), shows information on the client's credibility of payment. If the value is "1", the client is credible, "0" if not.

In [1]:

```
#numeric
import numpy as np
import pandas as pd
import xlrd
from pandas import DataFrame, read_csv
# graphics
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
#Disable warning
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)
from distutils.version import LooseVersion as Version
from sklearn import __version__ as sklearn_version
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, mutual_info_classif, f_classif
import numpy
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
if Version(sklearn_version) < '0.18':
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split
```

## Build table

In [2]:

```
df = pd.read_excel('default_credit_card_clients.xls')

# assign names to columns
df.columns = ['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_
2', 'PAY_3', 'PAY_4',
              'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT
4', 'BILL_AMT5', 'BILL_AMT6',
              'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_A
MT6', 'default payment next month']

df = df.drop(['ID'])

df
```

Out[2]:

| | LIMIT_BAL | SEX | EDUCATION | MARRIAGE | AGE | PAY_0 | PAY_2 | PAY_3 | PAY_4 | PAY |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 20000 | 2 | 2 | 1 | 24 | 2 | 2 | -1 | -1 | |
| **2** | 120000 | 2 | 2 | 2 | 26 | -1 | 2 | 0 | 0 | |
| **3** | 90000 | 2 | 2 | 2 | 34 | 0 | 0 | 0 | 0 | |
| **4** | 50000 | 2 | 2 | 1 | 37 | 0 | 0 | 0 | 0 | |
| **5** | 50000 | 1 | 2 | 1 | 57 | -1 | 0 | -1 | 0 | |
| **6** | 50000 | 1 | 1 | 2 | 37 | 0 | 0 | 0 | 0 | |
| **7** | 500000 | 1 | 1 | 2 | 29 | 0 | 0 | 0 | 0 | |
| **8** | 100000 | 2 | 2 | 2 | 23 | 0 | -1 | -1 | 0 | |
| **9** | 140000 | 2 | 3 | 1 | 28 | 0 | 0 | 2 | 0 | |
| **10** | 20000 | 1 | 3 | 2 | 35 | -2 | -2 | -2 | -2 | |
| **11** | 200000 | 2 | 3 | 2 | 34 | 0 | 0 | 2 | 0 | |
| **12** | 260000 | 2 | 1 | 2 | 51 | -1 | -1 | -1 | -1 | |
| **13** | 630000 | 2 | 2 | 2 | 41 | -1 | 0 | -1 | -1 | |
| **14** | 70000 | 1 | 2 | 2 | 30 | 1 | 2 | 2 | 0 | |
| **15** | 250000 | 1 | 1 | 2 | 29 | 0 | 0 | 0 | 0 | |
| **16** | 50000 | 2 | 3 | 3 | 23 | 1 | 2 | 0 | 0 | |
| **17** | 20000 | 1 | 1 | 2 | 24 | 0 | 0 | 2 | 2 | |
| **18** | 320000 | 1 | 1 | 1 | 49 | 0 | 0 | 0 | -1 | |
| **19** | 360000 | 2 | 1 | 1 | 49 | 1 | -2 | -2 | -2 | |
| **20** | 180000 | 2 | 1 | 2 | 29 | 1 | -2 | -2 | -2 | |
| **21** | 130000 | 2 | 3 | 2 | 39 | 0 | 0 | 0 | 0 | |
| **22** | 120000 | 2 | 2 | 1 | 39 | -1 | -1 | -1 | -1 | |
| **23** | 70000 | 2 | 2 | 2 | 26 | 2 | 0 | 0 | 2 | |
| **24** | 450000 | 2 | 1 | 1 | 40 | -2 | -2 | -2 | -2 | |
| **25** | 90000 | 1 | 1 | 2 | 23 | 0 | 0 | 0 | -1 | |
| **26** | 50000 | 1 | 3 | 2 | 23 | 0 | 0 | 0 | 0 | |
| **27** | 60000 | 1 | 1 | 2 | 27 | 1 | -2 | -1 | -1 | |
| **28** | 50000 | 2 | 3 | 2 | 30 | 0 | 0 | 0 | 0 | |
| **29** | 50000 | 2 | 3 | 1 | 47 | -1 | -1 | -1 | -1 | |
| **30** | 50000 | 1 | 1 | 2 | 26 | 0 | 0 | 0 | 0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **29971** | 360000 | 1 | 1 | 1 | 34 | -1 | -1 | -1 | 0 | |
| **29972** | 80000 | 1 | 3 | 1 | 36 | 0 | 0 | 0 | 0 | |
| **29973** | 190000 | 1 | 1 | 1 | 37 | 0 | 0 | 0 | 0 | |
| **29974** | 230000 | 1 | 2 | 1 | 35 | 1 | -2 | -2 | -2 | |

| | LIMIT_BAL | SEX | EDUCATION | MARRIAGE | AGE | PAY_0 | PAY_2 | PAY_3 | PAY_4 | PAY |
|---|---|---|---|---|---|---|---|---|---|---|
| **29975** | 50000 | 1 | 2 | 1 | 37 | 1 | 2 | 2 | 2 | |
| **29976** | 220000 | 1 | 2 | 1 | 41 | 0 | 0 | -1 | -1 | |
| **29977** | 40000 | 1 | 2 | 2 | 47 | 2 | 2 | 3 | 2 | |
| **29978** | 420000 | 1 | 1 | 2 | 34 | 0 | 0 | 0 | 0 | |
| **29979** | 310000 | 1 | 2 | 1 | 39 | 0 | 0 | 0 | 0 | |
| **29980** | 180000 | 1 | 1 | 1 | 32 | -2 | -2 | -2 | -2 | |
| **29981** | 50000 | 1 | 3 | 2 | 42 | 0 | 0 | 0 | 0 | |
| **29982** | 50000 | 1 | 2 | 1 | 44 | 1 | 2 | 2 | 2 | |
| **29983** | 90000 | 1 | 2 | 1 | 36 | 0 | 0 | 0 | 0 | |
| **29984** | 20000 | 1 | 2 | 1 | 44 | -2 | -2 | -2 | -2 | |
| **29985** | 30000 | 1 | 2 | 2 | 38 | -1 | -1 | -2 | -1 | |
| **29986** | 240000 | 1 | 1 | 2 | 30 | -2 | -2 | -2 | -2 | |
| **29987** | 360000 | 1 | 1 | 2 | 35 | -1 | -1 | -2 | -2 | |
| **29988** | 130000 | 1 | 1 | 2 | 34 | 0 | 0 | 0 | 0 | |
| **29989** | 250000 | 1 | 1 | 1 | 34 | 0 | 0 | 0 | 0 | |
| **29990** | 150000 | 1 | 1 | 2 | 35 | -1 | -1 | -1 | -1 | |
| **29991** | 140000 | 1 | 2 | 1 | 41 | 0 | 0 | 0 | 0 | |
| **29992** | 210000 | 1 | 2 | 1 | 34 | 3 | 2 | 2 | 2 | |
| **29993** | 10000 | 1 | 3 | 1 | 43 | 0 | 0 | 0 | -2 | |
| **29994** | 100000 | 1 | 1 | 2 | 38 | 0 | -1 | -1 | 0 | |
| **29995** | 80000 | 1 | 2 | 2 | 34 | 2 | 2 | 2 | 2 | |
| **29996** | 220000 | 1 | 3 | 1 | 39 | 0 | 0 | 0 | 0 | |
| **29997** | 150000 | 1 | 3 | 2 | 43 | -1 | -1 | -1 | -1 | |
| **29998** | 30000 | 1 | 2 | 2 | 37 | 4 | 3 | 2 | -1 | |
| **29999** | 80000 | 1 | 3 | 1 | 41 | 1 | -1 | 0 | 0 | |
| **30000** | 50000 | 1 | 2 | 1 | 46 | 0 | 0 | 0 | 0 | |

30000 rows × 24 columns

## Rank features (Univariate Feature Selection)

In [3]:

```
y=df.as_matrix(columns=[df.columns[23]])
X=df.as_matrix(columns=df.columns[1:])

#feature extraction (k=number of top features to select)
test = SelectKBest(f_classif, k=4)
fit = test.fit(X, y)

# summarized scores of each feature
numpy.set_printoptions(precision=3)
print("Scores of each feature (Credit Card): ")
print(fit.scores_)

features = fit.transform(X)

# print the selected features
print("\nSelected features (Credit Card): ")
print(features)
```

```
Scores of each feature (Credit Card):
[4.798e+01 2.355e+01 1.778e+01 5.789e+00 3.538e+03 2.239e+03 1.757e+
03
 1.477e+03 1.305e+03 1.085e+03 1.158e+01 6.044e+00 5.944e+00 3.095e+
00
 1.371e+00 8.658e-01 1.604e+02 1.033e+02 9.522e+01 9.719e+01 9.143e+
01
 8.509e+01       inf]

Selected features (Credit Card):
[[2 2 -1 1]
 [-1 2 0 1]
 [0 0 0 0]
 ...
 [4 3 2 1]
 [1 -1 0 1]
 [0 0 0 1]]
```

## Perform dimension reduction (PCA)

In [4]:

```
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=0)
y_train= y_train.astype('int')
y_train= y_train.flatten()
```

#### Standardize the data

In [5]:

```
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

#### Apply PCA

In [6]:

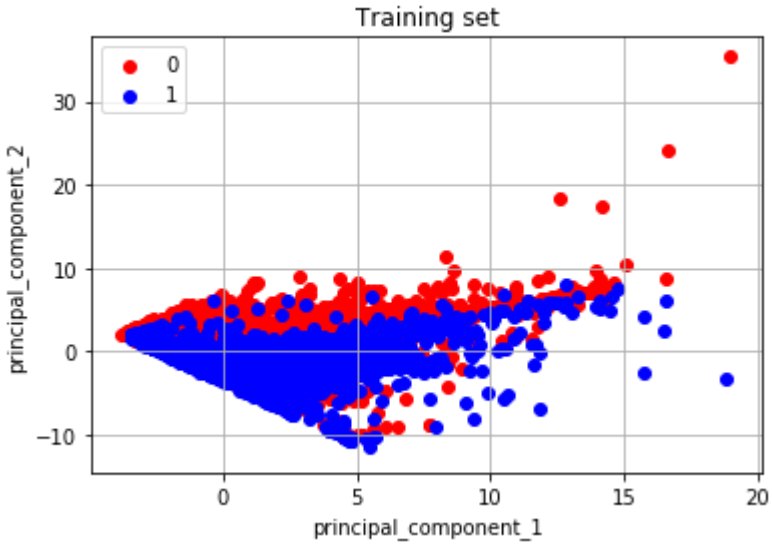
```
# Only two components
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)

print(X_train_pca)
```

```
[[-3.534  2.122]
 [ 0.483 -2.064]
 [-2.542  3.186]
 ...
 [ 0.785 -0.685]
 [-0.49  -0.773]
 [-0.536 -0.628]]
```

**Compute scatter plot**

In [7]:

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.figure
inx =(y_train==0)
inx=inx.ravel()
ax.scatter(X_train_pca[inx,0],X_train_pca[inx,1],marker='o',c='r', label='0')
inx =(y_train==1)
inx=inx.ravel()
ax.scatter(X_train_pca[inx,0],X_train_pca[inx,1],marker='o',c='b', label='1')
ax.set_title("Training set")
ax.set_xlabel('principal_component_1')
ax.set_ylabel('principal_component_2')
ax.legend()
ax.grid()
plt.show()
```



# Task B

On task B, we were tasked with:

1. Choosing two of the learning algorithms taught during the lectures and train the model
2. Tuning of the hyper-parameters used on the models

For this dataset, after analysing the previous plot, we decided that we would use only non-linear algorithms because:

- Our problem was not linearly separable (as we can see from the above plot).
- Our classification threshold cannot be a line, plane or hyperplane, like on linear methods.
- The dataset has high variance.

The algorithms chosen were two of the ones taught in classes: **SVM** (Support Vector Machine) [1] with rbf kernel and **MLP** (Multilayer Perceptron) [2].

Both algorithms perform binary classification and MLP can perform either classification or regression, depending upon its activation function. We chose "tanh" as the activation function because, in practice, it is usually preferred over Sigmoid function. Non-linear methods transform data to a new representational space (based on a kernel function) and then apply classification techniques to the data.

We obtained a fairly good separation of the two existing classes, on both MLP and SVM. Both models took a higher time to compute than on the linear algorithms, giving us more accurate results.

The two classes shown on the plots are separated by color, being red the value "0", which means "the client is not credible" on the original data and blue the value "1", which means "the client is credible" on the original data.

The algorithms used for classification require the assigment of hyper-parameters, whose values have influence on the performance of the model. In machine learning, hyper-parameter optimization or also known as "tuning" [5] is the search of a set of optimal hyper-parameters for a learning algorithm. With this is mind, we created two functions, **mlp_tunning** and **svc_tunning**, which use the search provided by the method **GridSearchCV** to exhaustively generate candidates from a grid of parameter values specified with the param_grid parameter [3]. The method uses the estimator API of scikit-learn. When “fitting” it on the dataset, all the possible combinations of parameter values are evaluated and the best combination is retained.

**Train the model with MLP and SVM algorithms**

In [8]:

```
#numeric
import numpy as np
import pandas as pd
# graphics
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from numpy import linalg as LA
from matplotlib.colors import ListedColormap
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from random import uniform
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# function taken from the class guide "Linear models for Classification"
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_m
in, 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

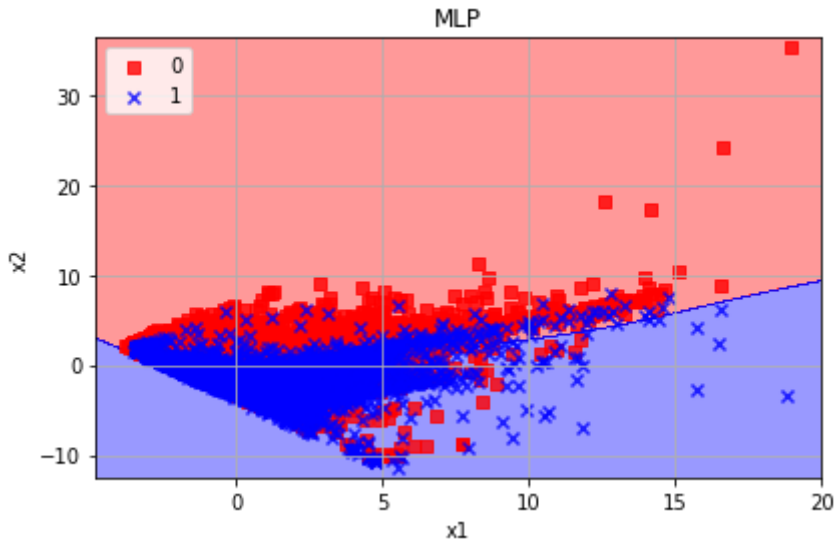
In [14]:

```
# function for tuning the hyper-parameters of the estimator
def mlp_tunning(X_train, y_train, clf):
    parameters = {
    'hidden_layer_sizes': [(100,)],
    'activation': ['tanh'],
    'alpha': [uniform(0.0001, 0.9)],
    'learning_rate': ['constant','adaptive'],
    'max_iter': [1000,2000,3000,4000,5000] }

    grid_search = GridSearchCV(estimator=clf, param_grid=parameters, n_jobs=-1,s
coring='accuracy', cv=5,)

    grid_search.fit(X_train, y_train)

    best_accuracy = grid_search.best_score_
    best_parameters = grid_search.best_params_

    print("Best accuracy (MLP): ", best_accuracy)
    print("Best parameters (MLP): ", best_parameters)

    return best_parameters

# uncomment to find the best hyper-parameters
#clf = MLPClassifier()
#best_parameters = mlp_tunning(X_train_pca,y_train, clf)
```

```
('Best accuracy (MLP): ', 0.8211428571428572)
('Best parameters (MLP): ', {'alpha': 0.634770521451485, 'activatio
n': 'tanh', 'max_iter': 2000, 'learning_rate': 'adaptive', 'hidden_l
ayer_sizes': (100,)})
```

In [16]:

```
mlp = MLPClassifier(activation='tanh', hidden_layer_sizes=(100,),alpha= 0.634770
521451485, learning_rate= 'adaptive', max_iter=2000)

a = np.array(y_train)
y_train = a.ravel()
y_train = y_train.tolist()

a = np.array(y_test)
y_test = a.ravel()
y_test = y_test.tolist()

mlp.fit(X_train_pca,y_train)

plot_decision_regions(X_train_pca, y_train, classifier=mlp)
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(loc='upper left')
plt.grid()
plt.title('MLP')
plt.tight_layout()
plt.show()
```




SVM

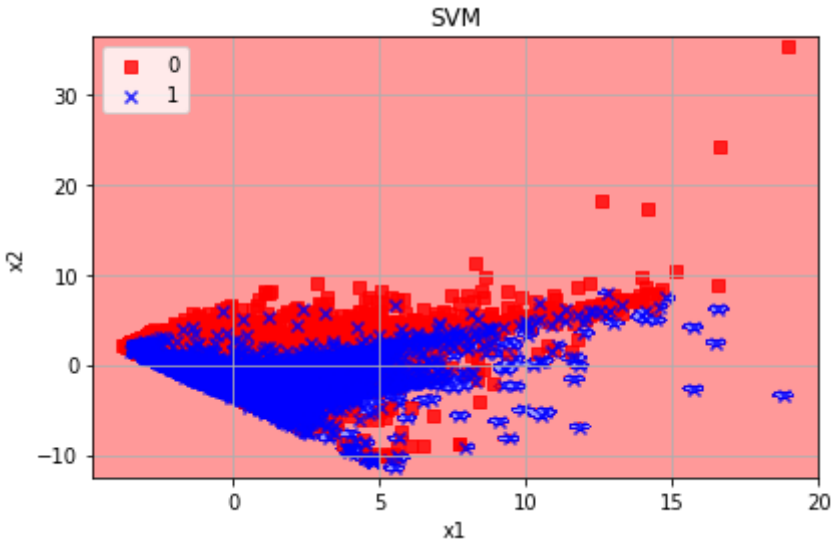
In [18]:

```
# function for tuning the hyper-parameters of the estimator
def svc_tunning(X_train, y_train, clf):

    parameters = [{'C': np.logspace(-3, 2, 6), 'kernel': ['rbf'],
                   'gamma': np.logspace(-3, 2, 6)}]

    grid_search = GridSearchCV(estimator=clf, param_grid=parameters, n_jobs=-1,
scoring='accuracy', cv=5,)

    grid_search.fit(X_train, y_train)

    best_accuracy = grid_search.best_score_
    best_parameters = grid_search.best_params_

    print("Best accuracy (SVM): ", best_accuracy)
    print("Best parameters (SVM): ", best_parameters)

    return best_parameters

# uncomment to find best hyper-parameters
#clf = SVC()
#best_parameters = svc_tunning(X_train_pca,y_train, clf)
```

```
('Best accuracy (SVM): ', 0.8674761904761905)
('Best parameters (SVM): ', {'kernel': 'rbf', 'C': 1.0, 'gamma': 10.
0})
```

In [19]:

```
svm=SVC(C=1.0,kernel='rbf',gamma=10.0)

svm.fit(X_train_pca,y_train)

plot_decision_regions(X_train_pca, y_train, classifier=svm)
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(loc='upper left')
plt.grid()
plt.title('SVM')
plt.tight_layout()
plt.show()
```




# Adult Dataset

The second dataset used was the one retrieved from the following link:
https://archive.ics.uci.edu/ml/datasets/adult (https://archive.ics.uci.edu/ml/datasets/adult)

# Task A

On Task A, the project stated that we had to follow the steps:

1. Mapping categorial data into numeric data
2. Rank features
3. Perform dimension reduction (Principal Component Analysis or Kernel Principal Components)

The data was loaded on the **Build table** section.

In this dataset, there was categorical data, which means that we had to map that same data into numeric data. That mapping was made with incremental integers.

The data was loaded on the **Build table** section. Then, on **Rank features (Univariate Feature Selection)**, we chose a univariate feature selection to find the scores (rank) of the best features, chosen by SelectKBest function.

Finally, we had to perform a dimension reduction on our data. For that, we used the well-known procedure **PCA**. After choosing X2 as all values of the table and Y2 as the column 15 (the output column), we split X2 and Y2 into random train and test subsets, standardized the data and only then we applied the PCA to the sets. The scatter plot was computed afterwards, showing the predicted values and with the 2 principal components in the axes.

The output column (last column on the .txt file), as information about wether the income of a person exceeds $50K/yr (">50K") or not ("<=50K").

In [20]:

```
#numeric
import numpy as np
import pandas as pd
import xlrd
from pandas import DataFrame, read_csv
# graphics
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
#Disable warning
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)
from distutils.version import LooseVersion as Version
from sklearn import __version__ as sklearn_version
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, mutual_info_classif
import numpy
```

## Build table

In [22]:

```
data = pd.read_csv('adult.data.txt', delimiter=", ", header=None, engine='pytho
n')

# assign names to columns
data.columns = ["age", "workclass", "fnlwgt", "education", "education-num", \
                "marital-status" ,"occupation", "relationship", "race", "sex" ,\
                "capital-gain", "capital-loss", "hours-per-week", "native-countr
y", \
                "probability-label"]
```

## Mapping categorical data into numeric data

In [23]:

```
workclass_map = {'Private': 1,'Self-emp-not-inc': 2,'Self-emp-inc': 3,'Federal-g
ov': 4,'Local-gov': 5,
                                 'State-gov': 6,'Without-pay': 7,'Never-worked': 8
}

education_map = {'Bachelors': 1,'Some-college': 2,'11th': 3,'HS-grad': 4,'Prof-s
chool': 5,
                                  'Assoc-acdm': 6,'Assoc-voc': 7,'9th': 8,'7th-8t
h': 9,'12th': 10,'Masters': 11,
                                  '1st-4th': 12,'10th': 13,'Doctorate': 14,'5th-6
th': 15,'Preschool': 16}

marital_map = {'Married-civ-spouse': 1,'Divorced': 2,'Never-married': 3,'Separat
ed': 4,
                                    'Widowed': 5,'Married-spouse-absent': 6,'Marr
ied-AF-spouse': 7}

occupation_map = {'Tech-support': 1,'Craft-repair': 2,'Other-service': 3,'Sales'
: 4,
                                 'Exec-managerial': 5,'Prof-specialty': 6,'Handle
rs-cleaners': 7,
                                 'Machine-op-inspct': 8,'Adm-clerical': 9,'Farmin
g-fishing': 10,
                                 'Transport-moving': 11,'Priv-house-serv': 12,'Pr
otective-serv': 13,
                                 'Armed-Forces': 14}

relationship_map = {'Wife': 1,'Own-child': 2,'Husband': 3,'Not-in-family': 4,
                                     'Other-relative': 5,'Unmarried': 6}

race_map = {'White': 1,'Asian-Pac-Islander': 2,'Amer-Indian-Eskimo': 3,'Other':
4,'Black': 5}

sex_map = {'Female': 1,'Male': 2}

country_map = {'United-States': 1,'Cambodia': 2,'England': 3,'Puerto-Rico': 4,
                                    'Canada': 5,'Germany': 6,'Outlying-US(Guam-US
VI-etc)': 7,'India': 8,
                                    'Japan': 9,'Greece': 10,'South': 11,'China':
12,'Cuba': 13,'Iran': 14,
                                    'Honduras': 15,'Philippines': 16,'Italy': 17,
'Poland': 18,'Jamaica': 19,
                                    'Vietnam': 20,'Mexico': 21,'Portugal': 22,'Ir
eland': 23,'France': 24,
                                    'Dominican-Republic': 25,'Laos': 26,'Ecuador'
: 27,'Taiwan': 28,'Haiti': 29,
                                    'Columbia': 30,'Hungary': 31,'Guatemala': 32,
'Nicaragua': 33,'Scotland': 34,
                                    'Thailand': 35,'Yugoslavia': 36,'El-Salvador'
: 37,'Trinadad&Tobago': 38,
                                    'Peru': 39,'Hong': 40,'Holand-Netherlands': 4
1}

probability_map = {'<=50K': 1, '>50K' : 2}

data['workclass'] = data['workclass'].map(workclass_map)
data['education'] = data['education'].map(education_map)
data['marital-status'] = data['marital-status'].map(marital_map)
data['occupation'] = data['occupation'].map(occupation_map)
```

```
data['relationship'] = data['relationship'].map(relationship_map)
data['race'] = data['race'].map(race_map)
data['sex'] = data['sex'].map(sex_map)
data['native-country'] = data['native-country'].map(country_map)
data['probability-label'] = data['probability-label'].map(probability_map)

#Drop rows with NaN values
data = data.dropna(how="any")
data
```

Out[23]:

| | age | workclass | fnlwgt | education | education-num | marital-status | occupation | relationship | rac |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 39 | 6.0 | 77516 | 1 | 13 | 3 | 9.0 | 4 | |
| **1** | 50 | 2.0 | 83311 | 1 | 13 | 1 | 5.0 | 3 | |
| **2** | 38 | 1.0 | 215646 | 4 | 9 | 2 | 7.0 | 4 | |
| **3** | 53 | 1.0 | 234721 | 3 | 7 | 1 | 7.0 | 3 | |
| **4** | 28 | 1.0 | 338409 | 1 | 13 | 1 | 6.0 | 1 | |
| **5** | 37 | 1.0 | 284582 | 11 | 14 | 1 | 5.0 | 1 | |
| **6** | 49 | 1.0 | 160187 | 8 | 5 | 6 | 3.0 | 4 | |
| **7** | 52 | 2.0 | 209642 | 4 | 9 | 1 | 5.0 | 3 | |
| **8** | 31 | 1.0 | 45781 | 11 | 14 | 3 | 6.0 | 4 | |
| **9** | 42 | 1.0 | 159449 | 1 | 13 | 1 | 5.0 | 3 | |
| **10** | 37 | 1.0 | 280464 | 2 | 10 | 1 | 5.0 | 3 | |
| **11** | 30 | 6.0 | 141297 | 1 | 13 | 1 | 6.0 | 3 | |
| **12** | 23 | 1.0 | 122272 | 1 | 13 | 3 | 9.0 | 2 | |
| **13** | 32 | 1.0 | 205019 | 6 | 12 | 3 | 4.0 | 4 | |
| **15** | 34 | 1.0 | 245487 | 9 | 4 | 1 | 11.0 | 3 | |
| **16** | 25 | 2.0 | 176756 | 4 | 9 | 3 | 10.0 | 2 | |
| **17** | 32 | 1.0 | 186824 | 4 | 9 | 3 | 8.0 | 6 | |
| **18** | 38 | 1.0 | 28887 | 3 | 7 | 1 | 4.0 | 3 | |
| **19** | 43 | 2.0 | 292175 | 11 | 14 | 2 | 5.0 | 6 | |
| **20** | 40 | 1.0 | 193524 | 14 | 16 | 1 | 6.0 | 3 | |
| **21** | 54 | 1.0 | 302146 | 4 | 9 | 4 | 3.0 | 6 | |
| **22** | 35 | 4.0 | 76845 | 8 | 5 | 1 | 10.0 | 3 | |
| **23** | 43 | 1.0 | 117037 | 3 | 7 | 1 | 11.0 | 3 | |
| **24** | 59 | 1.0 | 109015 | 4 | 9 | 2 | 1.0 | 6 | |
| **25** | 56 | 5.0 | 216851 | 1 | 13 | 1 | 1.0 | 3 | |
| **26** | 19 | 1.0 | 168294 | 4 | 9 | 3 | 2.0 | 2 | |
| **28** | 39 | 1.0 | 367260 | 4 | 9 | 2 | 5.0 | 4 | |
| **29** | 49 | 1.0 | 193366 | 4 | 9 | 1 | 2.0 | 3 | |
| **30** | 23 | 5.0 | 190709 | 6 | 12 | 3 | 13.0 | 4 | |
| **31** | 20 | 1.0 | 266015 | 2 | 10 | 3 | 4.0 | 2 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **32526** | 32 | 1.0 | 211349 | 13 | 6 | 1 | 11.0 | 3 | |
| **32527** | 22 | 1.0 | 203715 | 2 | 10 | 3 | 9.0 | 2 | |
| **32528** | 31 | 1.0 | 292592 | 4 | 9 | 1 | 8.0 | 1 | |
| **32529** | 29 | 1.0 | 125976 | 4 | 9 | 4 | 4.0 | 6 | |
| **32532** | 34 | 1.0 | 204461 | 14 | 16 | 1 | 6.0 | 3 | |

| | age | workclass | fnlwgt | education | education-num | marital-status | occupation | relationship | rac |
|---|---|---|---|---|---|---|---|---|---|
| **32533** | 54 | 1.0 | 337992 | 1 | 13 | 1 | 5.0 | 3 | |
| **32534** | 37 | 1.0 | 179137 | 2 | 10 | 2 | 9.0 | 6 | |
| **32535** | 22 | 1.0 | 325033 | 10 | 8 | 3 | 13.0 | 2 | |
| **32536** | 34 | 1.0 | 160216 | 1 | 13 | 3 | 5.0 | 4 | |
| **32537** | 30 | 1.0 | 345898 | 4 | 9 | 3 | 2.0 | 4 | |
| **32538** | 38 | 1.0 | 139180 | 1 | 13 | 2 | 6.0 | 6 | |
| **32540** | 45 | 6.0 | 252208 | 4 | 9 | 4 | 9.0 | 2 | |
| **32543** | 45 | 5.0 | 119199 | 6 | 12 | 2 | 6.0 | 6 | |
| **32544** | 31 | 1.0 | 199655 | 11 | 14 | 2 | 3.0 | 4 | |
| **32545** | 39 | 5.0 | 111499 | 6 | 12 | 1 | 9.0 | 1 | |
| **32546** | 37 | 1.0 | 198216 | 6 | 12 | 2 | 1.0 | 4 | |
| **32547** | 43 | 1.0 | 260761 | 4 | 9 | 1 | 8.0 | 3 | |
| **32548** | 65 | 2.0 | 99359 | 5 | 15 | 3 | 6.0 | 4 | |
| **32549** | 43 | 6.0 | 255835 | 2 | 10 | 2 | 9.0 | 5 | |
| **32550** | 43 | 2.0 | 27242 | 2 | 10 | 1 | 2.0 | 3 | |
| **32551** | 32 | 1.0 | 34066 | 13 | 6 | 1 | 7.0 | 3 | |
| **32552** | 43 | 1.0 | 84661 | 7 | 11 | 1 | 4.0 | 3 | |
| **32553** | 32 | 1.0 | 116138 | 11 | 14 | 3 | 1.0 | 4 | |
| **32554** | 53 | 1.0 | 321865 | 11 | 14 | 1 | 5.0 | 3 | |
| **32555** | 22 | 1.0 | 310152 | 2 | 10 | 3 | 13.0 | 4 | |
| **32556** | 27 | 1.0 | 257302 | 6 | 12 | 1 | 1.0 | 1 | |
| **32557** | 40 | 1.0 | 154374 | 4 | 9 | 1 | 8.0 | 3 | |
| **32558** | 58 | 1.0 | 151910 | 4 | 9 | 5 | 9.0 | 6 | |
| **32559** | 22 | 1.0 | 201490 | 4 | 9 | 3 | 9.0 | 2 | |
| **32560** | 52 | 3.0 | 287927 | 4 | 9 | 1 | 5.0 | 1 | |

30162 rows × 15 columns

## Rank features (Univariate Feature Selection)

In [24]:

```
array = data.values

X2 = array[:,0:15]
Y2 = array[:,14]

#feature extraction (k=number of top features to select)
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X2, Y2)

# summarized scores of each feature
np.set_printoptions(precision=3)
print("Scores of each feature (Adult): ")
print(fit.scores_)

features = fit.transform(X2)

print("\nSelected features (Adult): ")
print(features)
```

```
Scores of each feature (Adult):
[7.928e+03 2.723e+02 1.423e+05 7.424e+00 2.178e+03 2.888e+03 9.492e+
01
 4.313e+02 2.759e+02 1.852e+02 7.413e+07 1.256e+06 5.569e+03 1.349e+
03
 4.515e+03]

Selected features (Adult):
[[3.900e+01 7.752e+04 2.174e+03 0.000e+00]
 [5.000e+01 8.331e+04 0.000e+00 0.000e+00]
 [3.800e+01 2.156e+05 0.000e+00 0.000e+00]
 ...
 [5.800e+01 1.519e+05 0.000e+00 0.000e+00]
 [2.200e+01 2.015e+05 0.000e+00 0.000e+00]
 [5.200e+01 2.879e+05 1.502e+04 0.000e+00]]
```

## Perform dimension reduction (PCA)

In [25]:

```
from distutils.version import LooseVersion as Version
from sklearn import __version__ as sklearn_version
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
if Version(sklearn_version) < '0.18':
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split
```

### Prepare data for classification task

In [26]:

```
X_train, X_test, y_train, y_test = \
    train_test_split(X2, Y2, test_size=0.3, random_state=0)
y_train= y_train.astype('int')
y_train= y_train.flatten()
```

### Standardize the data

In [27]:

```
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

### Applying PCA

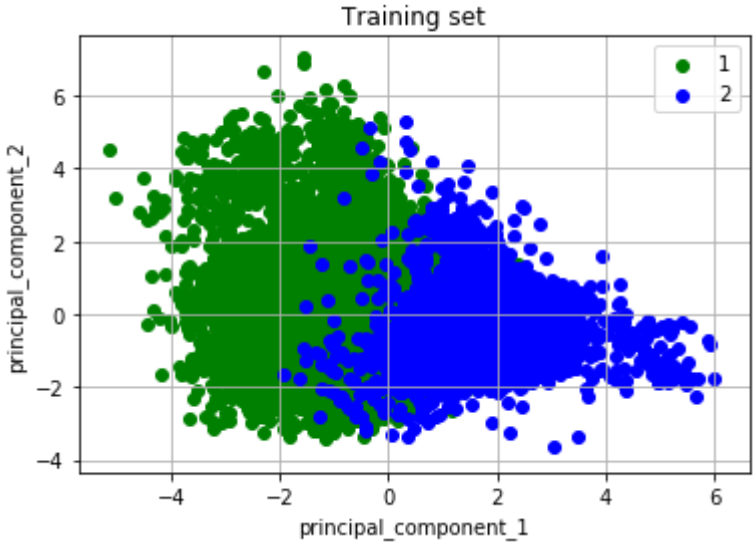
In [28]:

```
pca = PCA(n_components=2)
pc_X = pca.fit_transform(X_train_std)
pc_Xt = pca.transform(X_test_std)
```

### Compute plot

In [29]:

```
# label 1 = below 50K
# label 2 = above 50K
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.figure
inx =(y_train==1)
inx=inx.ravel()
ax.scatter(pc_X[inx,0],pc_X[inx,1],marker='o',c='g', label='1')
inx =(y_train==2)
inx=inx.ravel()
ax.scatter(pc_X[inx,0],pc_X[inx,1],marker='o',c='b', label='2')
ax.set_title("Training set")
ax.set_xlabel('principal_component_1')
ax.set_ylabel('principal_component_2')
ax.legend()
ax.grid()
plt.show()
```



# Task B

On task B, we were tasked with:

1. Choosing two of the learning algorithms taught during the lectures and train the model.
2. Tuning of the hyper-parameters used on the learning algorithms.

For this dataset, after analysing the previous plot, we decided that we would use only linear algorithms because:

- Linear methods can solve problems that are linearly separable (via a line, hyperplane...), as it is the case of our problem.
- Our classification threshold is approximately linear (a line).
- The dataset has low variance.

The algorithms chosen were two of the ones taught in classes: Perceptron [4] and SVM [1].

**SVM** and **Perceptron** are binary classifiers, making them the right choice for predicting the also binary output of this dataset. SVM is also a discriminative model, which means it tries to model by just depending on the observed data, while learning how to do the classification from the given statistics.

We obtained a fairly good separation of the two existing classes, and the algorithm took little time to compute, so we concluded that the linear algorithms were the best choice.

The predictions shown on the plots are separated by color, on both SVM and Perceptron, being green the value **1.0**, which means **<=50K** on the original data and blue the value **2.0**, which means **>50K** on the original data.

Taking into account the definition of hyper-parameters and tuning on the section Task B of the first data set, we created two functions, **svc_tunning** and **perceptron_tunning**, which use the search provided by the method **GridSearchCV** to exhaustively generate candidates from a grid of parameter values specified with the param_grid parameter [3]. When “fitting” the estimator on the dataset, all the possible combinations of parameter values are evaluated and the best combination is retained.

In [30]:

```
import numpy as np
import pandas as pd
# graphics
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from distutils.version import LooseVersion as Version
from sklearn import __version__ as sklearn_version
from numpy import linalg as LA
from matplotlib.colors import ListedColormap
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.linear_model import Perceptron
from random import uniform
```

### Train the model with SVM and Perceptron algorithms

In [38]:

```
# function taken from the class guide "Linear models for Classification"
def plot_decision_regions(X, y, classifier, resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('green','blue')
    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)
```

SVM

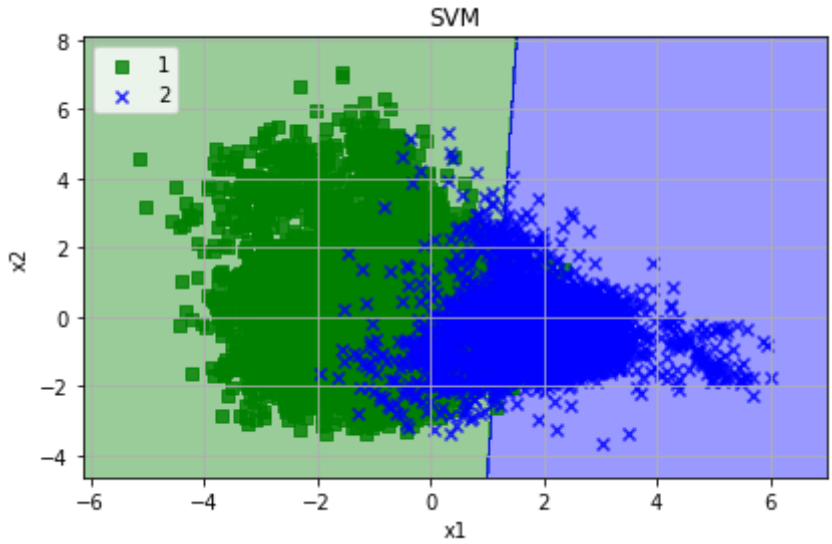
In [32]:

```
# function for tuning the hyper-parameters of the estimator
def svc_tunning(X_train, y_train,clf):
    parameters = [{'C': np.logspace(-3, 2, 6), 'kernel': ['linear'],
                   'gamma': np.logspace(-3, 2, 6)}]

    grid_search = GridSearchCV(estimator=clf, param_grid=parameters, n_jobs=-1,s
coring='accuracy', cv=5,)

    grid_search.fit(X_train, y_train)

    best_accuracy = grid_search.best_score_
    best_parameters = grid_search.best_params_

    print("Best accuracy (SVM): ", best_accuracy)
    print("Best parameters (SVM): ", best_parameters)

    return best_parameters

# uncomment to find best hyper-parameters
#clf = SVC()
#best_parameters = svc_tunning(pc_X,y_train,clf)
```

```
('Best accuracy (SVM): ', 0.9385686543835552)
('Best parameters (SVM): ', {'kernel': 'linear', 'C': 0.01, 'gamma':
0.001})
```

In [39]:

```
svm1=SVC(C= 0.01, gamma= 0.001, kernel= 'linear')
svm1

svm1=svm1.fit(pc_X,y_train)

plot_decision_regions(pc_X, y_train, classifier=svm1)
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(loc='upper left')
plt.grid()
plt.title('SVM')
plt.tight_layout()
plt.show()
```




Perceptron

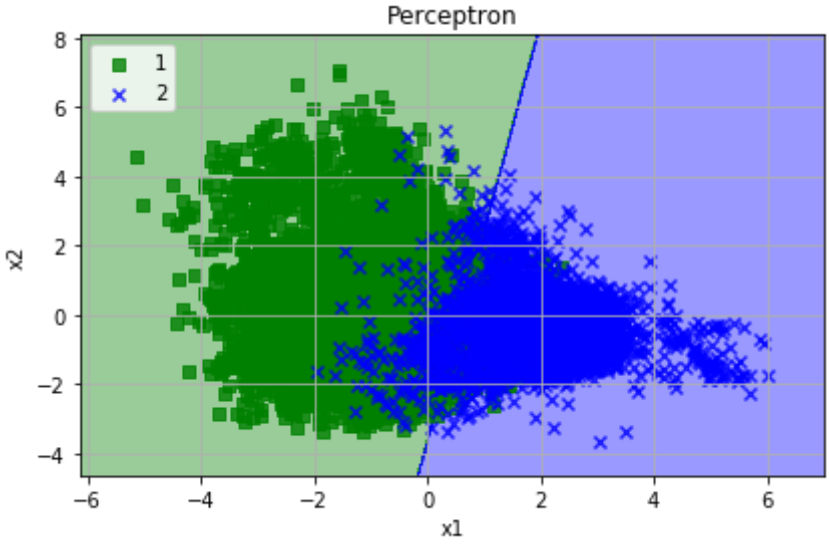
In [40]:

```
# function for tuning the hyper-parameters of the estimator
def perceptron_tunning(X_train, y_train, clf):

    parameters = [{'warm_start' : [False, True], 'penalty': ['l2'],
                'alpha': [uniform(0.0001, 0.9)]}]

    grid_search = GridSearchCV(estimator=clf, param_grid=parameters, n_jobs=-1,s
coring='accuracy', cv=5,)

    grid_search.fit(X_train, y_train)

    best_accuracy = grid_search.best_score_
    best_parameters = grid_search.best_params_

    print("Best accuracy (Perceptron): ", best_accuracy)
    print("Best parameters (Perceptron): ", best_parameters)

    return best_parameters

# uncomment to find best hyper-parameters
#clf = Perceptron()
#best_parameters = perceptron_tunning(pc_X,y_train, clf)
```

```
('Best accuracy (Perceptron): ', 0.5428882678918202)
('Best parameters (Perceptron): ', {'penalty': 'l2', 'alpha': 0.3381
541880242996, 'warm_start': False})
```

In [42]:

```
ppn=Perceptron(penalty='l2', alpha=0.3381541880242996, fit_intercept=True, eta0=
0.1, n_jobs=1, warm_start=False)

ppn.fit(pc_X,y_train)

plot_decision_regions(pc_X, y_train, classifier=ppn)
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(loc='upper left')
plt.grid()
plt.title('Perceptron')
plt.tight_layout()
plt.show()
```




# References

[1] https://scikit-learn.org/stable/modules/svm.html (https://scikit-learn.org/stable/modules/svm.html)

[2] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)

[3] https://scikit-learn.org/stable/modules/grid_search.html#exhaustive-grid-search (https://scikit-learn.org/stable/modules/grid_search.html#exhaustive-grid-search)

[4] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html)

[5] https://scikit-learn.org/stable/modules/grid_search.html (https://scikit-learn.org/stable/modules/grid_search.html)