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)

Complete project

On this notebook, we have the complete code of the project, which is also on notebook1. On notebook1, the code is explained in detail, but here only the code is displayed. On **Task C** section, we evaluate and discuss the performance of the learning algorithms used.

Credit Card Dataset

In [2]:

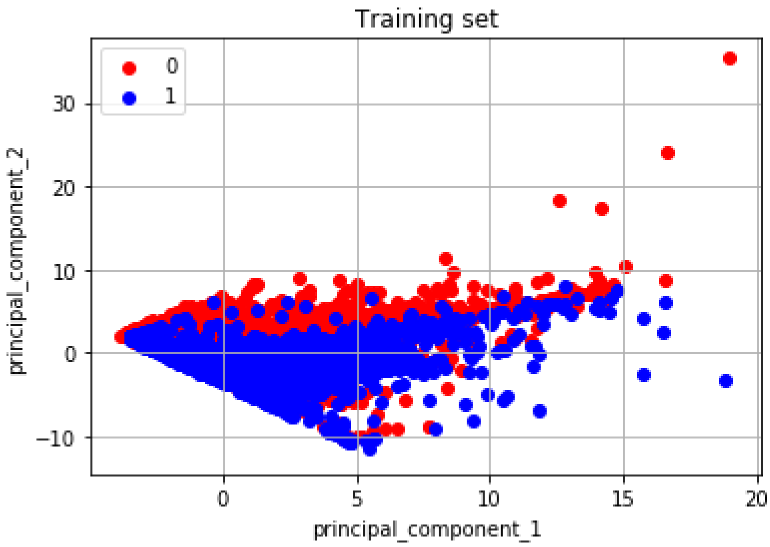
```
#numeric
import numpy as np
import pandas as pd
import xlrd
from pandas import DataFrame, read_csv
# graphics
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
#Disable warning
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", 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.preprocessing import StandardScaler
from sklearn.decomposition import PCA
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.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, mutual_info_classif, f_classif
import numpy

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

#### TASK A
#### Build table

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

#### Rank features (univariate feature selection)

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

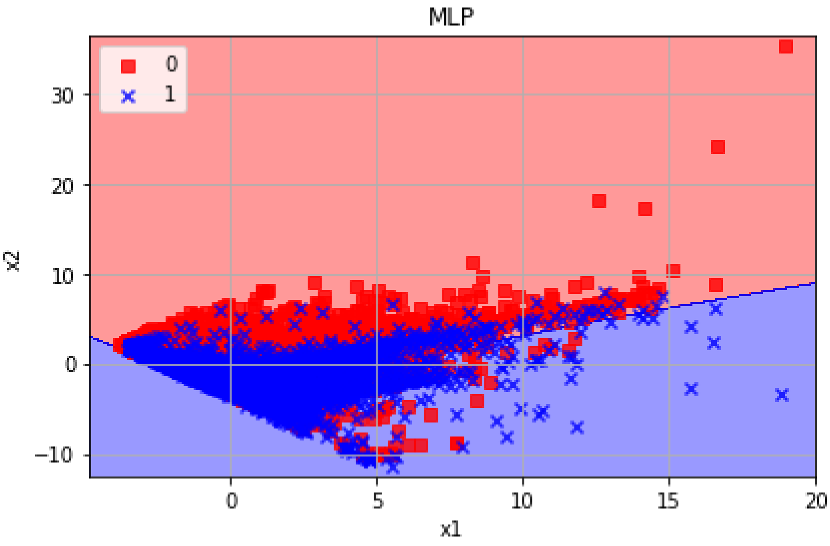
```python
#feature extraction (k=number of top features to select)
test = SelectKBest(f_classif, k=4)
fit = test.fit(X, y)

# 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("\nSelected features (Credit Card): ")
print(features)

#### Perform dimension reduction (PCA)

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

# apply PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)

# compute scatter plot
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
#### Training Model

# 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
```

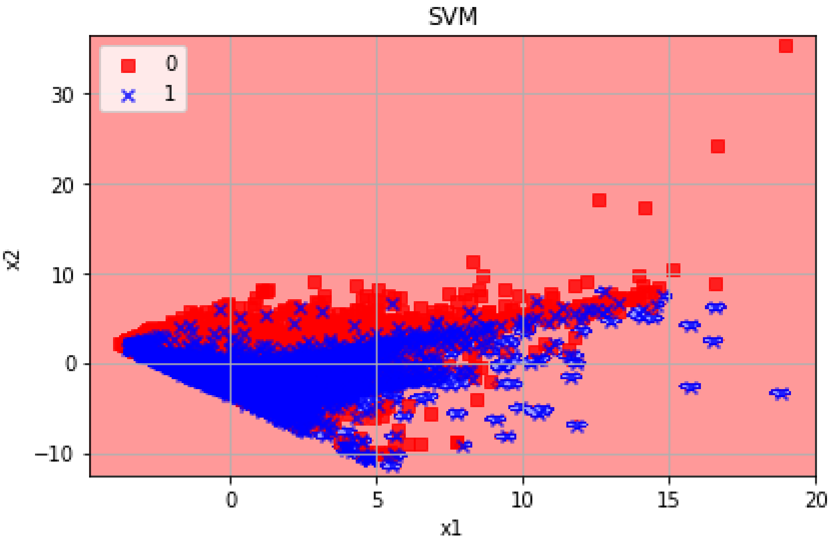
```
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution), np.arange(x2_m
in, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), 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
# 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)

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
# 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)

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()

#### TASK C
# function for evaluating the model
def eval_test(x_train, x_test, y_train, y_test, model, name):
    print("\n--Evaluate model {}--".format(name))
    print("K-fold CV accuracy mean (test set):", cross_val_score(model, x_train,
 y_train, scoring='accuracy', cv=10).mean())

    predicted_data = model.predict(x_train)
    print("Accuracy (training set):", accuracy_score(y_train, predicted_data))

    predicted_data = model.predict(x_test)
    print("Accuracy (test set):", accuracy_score(y_test, predicted_data))

eval_test(X_train_pca, X_test_pca, y_train, y_test, mlp, "MLP")
eval_test(X_train_pca, X_test_pca, y_train, y_test, svm,  "SVM")
```

```
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]]
```








```
--Evaluate model MLP--
('K-fold CV accuracy mean (test set):', 0.820522847435264)
('Accuracy (training set):', 0.821952380952381)
('Accuracy (test set):', 0.8238888888888889)

--Evaluate model SVM--
('K-fold CV accuracy mean (test set):', 0.86652308242074)
('Accuracy (training set):', 0.8861428571428571)
('Accuracy (test set):', 0.8744444444444445)
```

Adult Dataset

In [3]:

```
#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
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
    import numpy as np
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 sklearn.linear_model import Perceptron
from random import uniform
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
import sklearn


#### TASK A
#### Build table

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

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
```

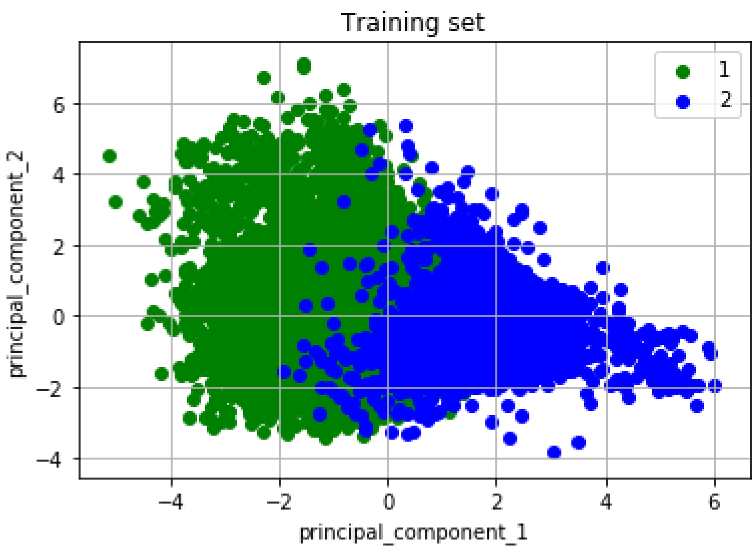
```
}

education_map = {'Bachelors': 1,'Some-college': 2,'11th': 3,'HS-grad': 4,'Prof-school': 5,
                                 'Assoc-acdm': 6,'Assoc-voc': 7,'9th': 8,'7th-8th': 9,'12th': 10,'Masters': 11,
                                 '1st-4th': 12,'10th': 13,'Doctorate': 14,'5th-6th': 15,'Preschool': 16}

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

occupation_map = {'Tech-support': 1,'Craft-repair': 2,'Other-service': 3,'Sales': 4,
                                'Exec-managerial': 5,'Prof-specialty': 6,'Handlers-cleaners': 7,
                                'Machine-op-inspct': 8,'Adm-clerical': 9,'Farming-fishing': 10,
                                'Transport-moving': 11,'Priv-house-serv': 12,'Protective-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-USVI-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,'Ireland': 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': 41}

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)
```

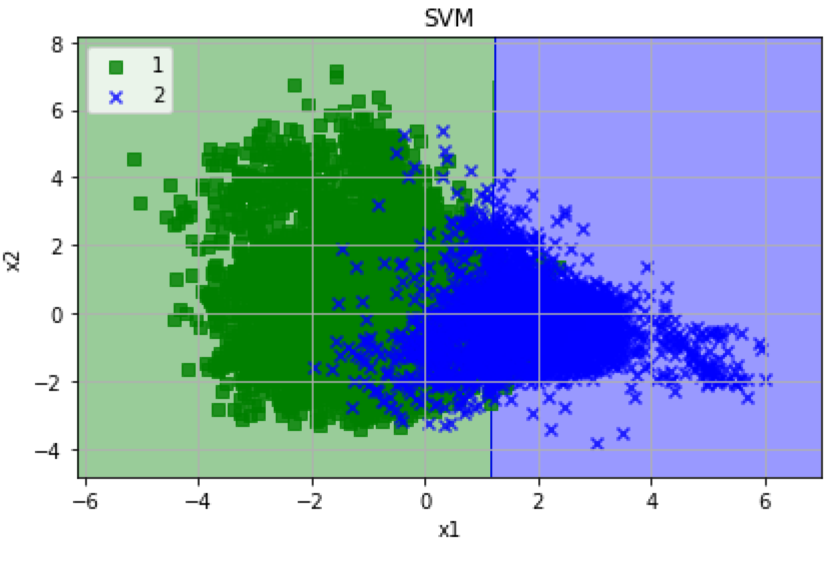
```
data = data.dropna(how="any")

#### Rank features (Univariate Feature Selection)

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)

#### Perform dimension reduction (PCA)

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

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

# compute plot
# 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
```

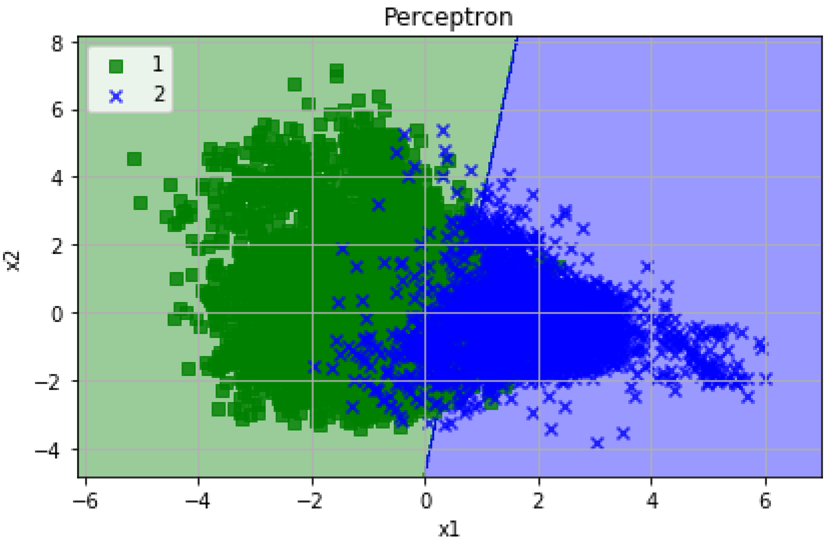
```
#### Training Model

# 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
# 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)

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.title('SVM')
plt.grid()
plt.tight_layout()
plt.show()

# Perceptron
```

```
# 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)

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()

#### TASK C
# function for evaluating the model
def eval_test(x_train, x_test, y_train, y_test, model, name):

    print("\n--Evaluate model {}--".format(name))
    print("K-fold CV accuracy mean (test set):", cross_val_score(model, x_train,
 y_train, scoring='accuracy', cv=10).mean())

    predicted_data = model.predict(x_train)
    print("Accuracy (training set):", accuracy_score(y_train, predicted_data)) #
y_true/y_pred

    predicted_data = model.predict(x_test)
    print("Accuracy (test set):", accuracy_score(y_test, predicted_data))

eval_test(pc_X, pc_Xt, y_train, y_test, svm1, "SVM")
eval_test(pc_X, pc_Xt, y_train, y_test, ppn, "Perceptron")
```

```
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]]
```








```
--Evaluate model SVM--
('K-fold CV accuracy mean (test set):', 0.9390896620157043)
('Accuracy (training set):', 0.939279117131625)
('Accuracy (test set):', 0.9345784064537518)

--Evaluate model Perceptron--
('K-fold CV accuracy mean (test set):', 0.7876624337883811)
('Accuracy (training set):', 0.88973618149955)
('Accuracy (test set):', 0.8865067963310863)
```

Task C

We were tasked with evaluating the performance of the learning algorithms used on our datasets. Both datasets were, then, chosen for the evaluation.

We chose to perform a K-fold cross-validation for estimating the performance of the learning algorithms:

- Support Vector Machine with rbf kernel
- Multilayer Perceptron
- Support Vector Machine with linear kernel
- Perceptron

The evaluation of the training models' performance was obtained by executing the code under the python comment "Task C".

About cross-validation

Cross-validation[1] or, in this case, K-fold cross-validation is a method that splits randomly a dataset into ‘k’ partitions [2]. One of those partitions is used as a test set and the rest are used as a training set. The model is then trained on the training sets and scored on the test set. Then the process is repeated until each unique partition as been used as the test set.

Our approach and results

The evaluation of our models has 3 components: 'K-fold CV accuracy mean', which is the mean of the accuracy values of cross validation, the 'Accuracy (training set)', which is the accuracy of the model applied to the whole training set (no folds) and finally the 'Accuracy (test set)', which is also the accuracy of the model, but applied to the whole test set (no folds).

To evaluate the performance of our models using cross-validation, we used the function **cross_val_score()** of sklearn. Its parameter 'cv' (cross-validation value [3]) is equal to '10', being cv the number of folds (partitions or sets) to be used. The scoring parameter was also defined, being equal to "accuracy", because that is the metric we wanted to evaluate. Accuracy [6] is then, the return of the function and it is the ratio of correctly predicted observations to the total observations [7].

Our choice of number of folds allows the size of each partition to be large enough to provide a fair estimate of the model’s performance on it, taking into account its size. At the same time, it is not too small, such that we don’t have enough trained models to evaluate and with this value, there is no costly computation.

We decided to use cross-validation instead of the hold-out method, because cross-validation gives our models the "opportunity" to train on multiple partitions. Then, after calculating each score (accuracy) of the models and their mean, we had a better view on how well that same model would perform if it was applied on new data ("unseen" data) [4]. Hold-out, on the other hand, is dependent on only one train-test partition, which makes its score (accuracy) dependent on how the data is split into the train and test sets [5].

We also decided to compute the accuracy on one partition, as stated before, the training set and test set, to compare the results of accuracy between that scheme and the K-fold.

After analysing the recorded accuracy score of each of the models, we compiled all the information on the following table (the results are also shown on the output of the code):

Algorithm	K-fold CV accuracy mean	Accuracy (training set)	Accuracy (test set)
MLP	0.820522847435264	0.821952380952381	0.8238888888888889

Algorithm	K-fold CV accuracy mean	Accuracy (training set)	Accuracy (test set)
SVM (rbf)	0.86652308242074	0.8861428571428571	0.8744444444444445
SVM (linear)	0.9390896620157043	0.939279117131625	0.9345784064537518
Perceptron	0.7876624337883811	0.88973618149955	0.8865067963310863

The accuracy (train set) and accuracy (test set) were very similar within the same model, and on the contrary to our expectations, the values obtained with one partition or several (K-fold) resulted in approximately the same accuracy.

All computed accuracies were above 0.80 (Perceptron was an exception), very close to 1, which is a fairly positive result, meaning all our models would be able to generalize, with a good performance, to new input data (no overfitting [4]), specially the SVM (linear) algorithm, which obtained the best score.

About used tools

To run our code, it is necessary to have the following tools installed:

- pandas (https://pandas.pydata.org/pandas-docs/stable/install.html (https://pandas.pydata.org/pandas-docs/stable/install.html))
- scikit-learn (https://scikit-learn.org/stable/install.html (https://scikit-learn.org/stable/install.html))
- xlrd (https://pypi.org/project/xlrd/ (https://pypi.org/project/xlrd/))
- numpy (https://scipy.org/install.html (https://scipy.org/install.html))

References

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

[2] https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a (https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a)

[3] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html)

[4] https://elitedatascience.com/overfitting-in-machine-learning (https://elitedatascience.com/overfitting-in-machine-learning)

[5] https://medium.com/@eijaz/holdout-vs-cross-validation-in-machine-learning-7637112d3f8f (https://medium.com/@eijaz/holdout-vs-cross-validation-in-machine-learning-7637112d3f8f)

[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html)

[7] https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/ (https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/)