— Apply linear SVM on synthetic and real datasets
— Investigate the choice of hyper-parameter C
— Extend to multi-class classification problem
— *Provided codes* : functions included in `utility_svm.py` on Moodle.

This Session relies on the same materials as for Logistic Regression Session. You should refer to it for data loading, processing and for comparison purpose.

1 Synthetic data

Let consider a binary classification problem with class $y \in \{0, 1\}$. The samples of each class k are drawn according to a gaussian distribution with mean μ_k and covariance matrix $\mathbf{S}_k$. We will use the SKLearn SVM implementation through out this session for efficiency purpose.

1. Generate $n_1 = n_2 = 300$ training samples per class using $\mu_1 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$, $\mathbf{S}_1 = \frac{3}{2}\begin{pmatrix} 1 & 0.1 \\ 0.1 & 1 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} -2 \\ -2 \end{pmatrix}$ and $\mathbf{S}_2 = \frac{7}{2}\begin{pmatrix} 1 & -0.25 \\ -0.25 & 1/2 \end{pmatrix}$

```
from utility_svm import gen_data_twogaussians_2d
import numpy as np
# class 1
n1 = 300
mu1 = np.array([0, 2]); S1 = 1.5*np.array([[1, 0.1], [0.1, 1]])
# class 2
n2 = n1
mu2 = np.array([-2, -2]); S2 = 3.5*np.array([[1, -0.25], [-0.25, 1/2]])
Xtrain, Ytrain = gen_data_twogaussians_2d(mu1, S1, mu2, S2, n1, n2)
```

2. Split the data into respectively training, validation and test sets. Why do we need this data splitting ? Why do we need to shuffle and stratify the data for the splitting ?

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, Ytrain, Ytest = train_test_split(Xtrain, Ytrain, shuffle=
    True, test_size=1/3, stratify=Ytrain)
Xtrain, Xval, Ytrain, Yval = train_test_split(Xtrain, Ytrain, shuffle=
    True, test_size=1/2, stratify=Ytrain)
```

   Check the dimensions of each set.

3. (a) Learn a linear SVM using the training set.

```
from sklearn.svm import SVC
# define a linear SVM with C = 1
C = 1
clf_svm = SVC(kernel="linear", C = C) # we seek a linear svm (
    kernel ="linear")
# fit the parameters of the model
clf_svm.fit(Xtrain, Ytrain)
```

(b) Plot the decision frontier. Comment on the results.

```
# plot the decision boundary
import matplotlib.pyplot as plt
from utility_svm import plot_decision_regions_2d
plot_decision_regions_2d(Xtrain, Ytrain, clf_svm, resolution=0.02,
    title="Linear SVM")
```

The support vectors and their number are respectively provided in `clf_svm.n_support_` and `clf_svm.support_vectors_`. How many support vectors do we have ? Highlight these samples on the plot.

(c) What is the validation error rate ?

```
from sklearn.metrics import accuracy_score
val_err_rate = 1 - accuracy_score(Yval, clf_svm.predict(Xval))
print(...)
```

Compare it to the training error rate.

4. Change the value of C and comment on how the decision frontier and the margin behave.

```
vectC = np.logspace(-3, 2, 6) #vector of C in logarithmic scale
for C in vectC:
  clf_svm.C = C
  clf_svm.fit(Xtrain, Ytrain)
  plot_decision_regions_2d(Xtrain, Ytrain, clf_svm, resolution=0.02,
      title="Linear SVM with c = {}".format(C))
```

5. We want to select the best value of C in the logarithmic range $\{10^{-3}, \cdots, 10^{2}\}$ by checking the validation error rate. To do so, for each value of C, train a SVM model, compute and store its error rate on the validation set. For sanity check purpose we will also compute the training error rate.

```
# Select C by cross-validation
vectC = np.logspace(-3, 2, 10)
val_err_rate = np.empty(vectC.shape[0])
train_err_rate = ...
for ind_C, C in enumerate(vectC):
  # learn the SVM for C
  ...
  # compute the classification error rates
  val_err_rate [ind_C] = 1 - accuracy_score(Yval, clf_svm.predict(Xval))
  train_err_rate[ind_C] = ...
```

Plot the obtained error curves. What is the optimal value C_{opt} to select ?

```
# Error curves
plt.figure()
plt.semilogx(vectC, train_err_rate, "bs--", label="Training")
plt.semilogx(vectC, val_err_rate, "go--", label="Validation")
plt.xlabel("Parameter C")
plt.ylabel("Error rates")
plt.legend(loc="best")
```

Let select C_{opt}. Is it safe and sound to select C_{opt} on the training error rate basis ?

```
ind_min = val_err_rate.argmin()
Copt = vectC[ind_min]
print("\n Optimal C = {}".format(Copt))
```

6. Train the optimal SVM corresponding to C_{opt} ane evaluate its performance on the training, validation and test sets. Discuss the obtained results.

```
# Fit the optimal SVM using Copt
...
# compute and print the performances on train, validation and test sets
err_test = ...
print("Optimal Linear SVM : test error rate = {}".format(err_test))
...
```

2 Spam classification

We intend to spam classification problem using a linear SVM. We recall that the dataset contains 4601 e-mails, from which 57 features have been extracted (refer to previous Session for the details). The dataset `spambase.data` and the features name `spambase_variables.csv` are in text format and available on Moodle.

1. Read the files and extract the inputs X and the output Y (last column in the dataset). Refer to logistic regression labwork for the details.
2. Split the data into training and test sets. The test set size should be $1/3$ of the data.
3. Our goal is to learn a linear SVM able to classify the e-mails.
   (a) As we have seen in the first exercise, the SVM classifier requires to properly tune the C-parameter. For the sake, split your current training set in halves (respectively the new training set and a validation one).

```
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train
    , shuffle=True, test_size=1/2, stratify=Y_train)
```

   (b) Normalize the data by centering and scaling the variables.

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler(with_mean=True, with_std=True)
sc = sc.fit(X_train)
X_train = sc.transform(X_train)
X_val = sc.transform(X_val)
X_test = sc.transform(X_test)
```

   Is it the convenient way to normalize the datasets ?

   (c) Learn a linear SVM model. You should highlight how the optimal C^* parameter is selected and how the related final SVM model is learned. Hint : you can look for C in the range $\{10^{-3}, \cdots, 10^2\}$.

(d) Evaluate the accuracy performances of your SVM. Compare to your previous results using logistic regression model.

3 Multi-class SVM : digits classification

The goal is to apply mult-class SVM to solve the digit classification problem using MNIST dataset. Recall that the training and test data-sets respectively contain 60,000 and 10,000 gray images of the digits $0-9$. They are of dimension 28×28 vectorized into vectors of size 784.

1. Load the datasets

```
import numpy as np
# training set
mnist_train = np.loadtxt("./mnist/mnist-app.csv", delimiter=",")
Y_train = mnist_train[:,-1]
X_train = mnist_train[:, 0:784]
# test set
mnist_test = np.loadtxt("./mnist/mnist-test.csv", delimiter=",")
Y_test = mnist_test[:,-1]
X_test = mnist_test[:, 0:784]
```

2. To lower the computation cost we will only retain three digits at your choice and extract only samples corresponding to those digits. For instance let select digits $0, 2$ and 8 (feel free to change them).

```
digits = [0, 2, 8]
# squeeze the training to the selected digits
index_train = np.argwhere((Y_train == digits[0]) | (Y_train == digits
    [1]) | (Y_train == digits[2]))
X_train = X_train[np.squeeze(index_train), :]
Y_train = Y_train[np.squeeze(index_train)]
# squeeze the test set as well
index_test = np.argwhere((Y_test == digits[0]) | (Y_test == digits[1]) |
     (Y_test == digits[2]))
X_test = X_test[np.squeeze(index_test), :]
Y_test = Y_test[np.squeeze(index_test)]
```

3. Inspiring from the previous exercises learn a linear multi-class SVM. Which strategy is used to solve the multi-class problem ? Hint : check the documentation of `sklearn.svm.SVC`. How many support vectors does your final SVM model have ?