SVM

Import packages:

```
import pandas as pd
import numpy as np

from sklearn import svm

import matplotlib.pyplot as plt
```

Generate example data:

```
n_cls = 20   # Number of samples in each class.

np.random.seed(1)  # for reproducibility
x11 = np.random.normal(0.5, 1, (n_cls, 1))
np.random.seed(2)  # for reproducibility
x12 = np.random.normal(0.4, 1, (n_cls, 1))
np.random.seed(3)  # for reproducibility
x21 = np.random.normal(-0.3, 1, (n_cls, 1))
np.random.seed(4)  # for reproducibility
x22 = np.random.normal(-0.5, 1, (n_cls, 1))

X = np.vstack((
    np.hstack((x11, x12)),
    np.hstack((x21, x22))
))

y = np.hstack((-1 * np.ones(n_cls), +1 * np.ones(n_cls)))
```

Remember that before running a function with a random generator (for example: `np.random.normal()`) you have to manualy set a generator seed (with `np.random.seed()` function) – this way results of the experiments will be repeatable.

Function `np.vstack()` "joins" matrices along rows, and `np.hstack()` – along columns.

Train a model of a linear SVC classifier (with regularisation coeficient $C = 1000$):

```
clf = svm.SVC(kernel='linear', C=1000, random_state=1, probability=True)
clf.fit(X, y)
```

Notice, that while initialising an `svm.SVC` class object we pass an argument `random_state`, so that an object will not be using its own random state generator with our own seed – this way the results will be repeatable.

When using a `probability=True` argument an SVC model will also calculate probabilities *a posteriori* for the predicted classes.

Visualise feature space, observations, support vectors and decision boundaries with margin:

```
fig = plt.figure(1)
plt.clf()

plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
# plot support vectors
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='none', edgecolors='k')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Data')
plt.show()
```

Make classification of new samples:

```
X_new = np.array([
    [1, -0.4],
    [-1, -0.85],
    ])

y_new = clf.predict(X_new)

plt.scatter(X_new[:, 0], X_new[:, 1], c=y_new, s=100)
fig.canvas.draw()
fig.canvas.flush_events()
```

Calculate probabilities *a posteriori* for new samples:

```
# Get posterior probabilities for predictions (requires SVC to be created
with
#   `probability=True`).
post_probs = clf.predict_proba(X_new)
```

```
print(post_probs)
```

Zadanie – Klasyfikacja "Czy dany e-mail to spam?"

Konspekt:

svm_spam.pdf

Szkielet programu:

svm_spam_skeleton.zip

pakiet `nltk`:

nltk.zip

(wypakuj archiwum do folderu projektu, jeśli nie masz możliwości zainstalowania `nltk` z użyciem `pip`/`conda`)

Wczytywanie e-maili

While implementing a `read_file()` function you can look for help in the folowing tutorial Reading and Writing Files (PyDoc).

Wczytywanie listy słów

While implementing `get_vocabulary_dict()` function you can follow the example from `csv.reader()`.
To do a text conversion of a number saved as character string `s` into an integer use `int(s)`.

Przetwarzanie e-maila

To change string into lower letters use `str.lower()`.

To change strings into tokens use `re` (see `re.sub()`).

Useful elements:

- `[…]` – matching to any character listed in parentheses (for example `[a2]` – small letter A or number 2)
- `[x-y]` – matching any character from x to y (for example `[a-z]` – any small letter)
- X+ – matching to one or more instances of the pattern X (for example `[a2]+` will adjust a, aa, 2, 2a2 etc.)
- \S – matching to any non-white character (for example space etc.)
- `(X|Y)` – adjusting a pattern X or pattern Y
- X* – matching any number of instances of the X pattern,including zero times (for example `(ab)*` will adjust to an empty string ab, abab etc.)

Converting e-maila into feature vector

Useful functions:

- `numpy.zeros()`
- `numpy.reshape()` (see parameter newshape or `-1` dimension)

Evaluating the classifier

To calculate the average accuracy use `numpy.mean()` and logical indexing.

Analysis of an SVC model

Weights of an SVC model are stored in `coef_` atribute of a class `sklearn.svm.SCV`.

To determine the decreasing order of weights, use the function `numpy.argsort()`.