Practical task 8.

Analysis of SVM decision rule for model task

To complete this task you're asked to implement the following functions:

```
learn_stat(train_data, train_labels, test_data, test_labels, kernel, C):
(train_score, test_score, support)
```

This function trains SVM with specified kernel and the cost of misclassification C. The function returns train and test set classification accuracies and the number of support vectors in the decision rule.

The arguments:

1. `train_data` - numpy array of shape $N \times D$, storing train objects' features in rows.
2. `train_labels` - numpy array of shape N, storing train labels.
3. `test_data` - numpy array of shape $M \times D$, storing test objects' features in rows.
4. `test_labels` - numpy array of shape M, storing test labels.
5. `kernel` - the kernel.
6. `C` - cost of misclassification.

The return values:

1. `train_score` - accuracy on train set
2. `test_score` - accuracy on test set
3. `support` - number of support vectors in the decision rule

Useful functions and classes: sklearn.svm.SVC

```
plot_stat(train_data, train_labels, test_data, test_labels, kernel, minC, maxC, steps)
```

This function plots figures of train, test accuracies and the number of support vectors as functions of C, using the function `learn_stat`. All points of figures must be in logarithmic scale of C.

The arguments:

1. Look at the arguments of `learn_stat`.
2. `minC, maxC` - minimal and maximal values of C, defining the range for figures.
3. `steps` - number of points in generated range of C values.

Useful functions and classes: numpy.logspace

```
plot_support(train_data, train_labels, test_data, test_labels, kernel, C)
```

This function visualizes the separating hyperplane, acquired in the training process with the defined kernel and cost of misclassification. This function also visualizes the support vectors for each class (highlighted differently for each class) as well as objects of test set. **Additionally it visualizes hyperplanes that correspond to margin equal to 1 for each class.**

The arguments:

1. Look at the arguments of function `learn_stat`.

Useful functions and classes: numpy.meshgrid, pyplot.contourf.

The task:

1. Load the train and test sets from files `synth_train.csv` and `synth_test.csv` respectively.
2. Using the function `plot_stat`, analyze the decision rule for linear and RBF kernels. The values of C must be in the logarithmic scale in the range of $[10^{-2}, 200]$ with at least 30 steps.
   (a) What conclusions can you draw from the resulting figures?
   (b) What is common and what is distinct between classifiers with two different kernels?
   (c) Is there a connection between train and test accuracies?
   (d) Can one use the train accuracy to find the optimal value of C? What is another way of finding the optimal value of C without access to test set?
3. Using the function `plot_support`, analyze the decision rules acquired in the training process of SVM with linear and RBF kernels for three values of C: inadequately small, inadequately huge and optimal (choose the best value using the test set accuracy). What are the diffrencies between decision rules for different values of C and different kernels? **Explain why support vectors are located like this.**

Implementing kernel for strings

In this task you are asked to train a classifier, that detects the language (English or French), to which the input word belongs. Taking into account that words do not have a canonical numerical representation (that could be used for SVM classification), you must define a custom kernel, that computes the distance between two arbitrary strings. For training and testing use the words from files `en.txt` and `fr.txt`. Be carefull with first words in these files — they are in English in both files. Take from 100 to 400 words for training and 300 words for testing. Try to get at least 0.6 accuracy. Analyze the train and test accuracies for different values of C.

Hints:

1. to find the distance between two strings you can use the Levenshtein distance. You can implement the function for it by yourself or use python-Levenshtein package.
2. Constructing a custom kernel using Levenshtein distance you should remember that kernel function must satisfy some conditions. Therefore it is better to use some standard kernel that depends on the distance between objects.
3. the class SVC treats the objects of train and test sets as numeric arrays (even if you define a custom kernel). To overcome this issue, supply indices of objects in the `fit` and `score` methods. Your kernel function must use indices of the objects to calculate the distance between them (these indeces refer to some external list with words).
4. The class SVC assumes that your kernel is a function accepting two matrices of samples, X and Y (during training both of them are equal to training data matrix and during testing one of them is train matrix and another is test matrix) and you should return a matrix G where $G_{ij} = K(X_i, Y_j)$ — kernel function for pair of objects X_i, Y_j. See an example in the answer here.