CS 422/622 Extra Project 5

Due 12/8 11:59pm

Logistics: You must implement everything stated in this project description that is marked with an **implement** tag. Do not forget to include a README for the project. **You may import numpy, and matplotlib. No late work will be accepted.** I advise that you submit to Webcampus regularly and well before the deadline.

Deliverables: You should submit a single ZIP file, containing your project code (svm.py file) and your writeup (PDF). 622 students must use LaTeX for their write-up. Your zip file should be named firstname_lastname_project5.zip. For example, a zip file might look like emily_hand_project5.zip. Your code should run without errors on the ECC linux machines. If your code does not run for a particular problem, you will lose 50% on that problem. **You will lose 10 points for an incorrectly named file or incorrectly named functions.** *CHECK YOUR SPELLING.* The file for each section should have all functions implemented, such that we can *import your file and run your functions.* **If you submit code that plots/prints things, you will be docked points.**

1 Binary Classification (50 points)

You will implement a linear classifier with a decision boundary that maximizes the margin between positive and negative samples (SVM). The details of SVM can be quite complex, and so here you will implement a simpler version of it for 2D data.

We provide you with a function for generating data (features and labels):

```
data = generate_training_data_binary(num)
```

`num` indicates the training set. There are four different datasets `(num=1, num=2, num=3, num=4)` which you will need to train on and provide results for. All the datasets are linearly separable. `data` is a numpy array containing 2D features and their corresponding label in each row. So `data[0]` will have the form `[x1, x2, y]` where `[x1, x2]` is the feature for the first training sample and it has the label `y`. `y` can be -1 or 1.

We have also provided you with a function to plot the training data so you can visualize it:

```
plot_training_data_binary(data)
```

Implement: You must write the following function:

```
[w,b,S] = svm_train_brute(training_data)
```

This function will return the decision boundary for your classifier and a numpy array of support vectors `S`. That is, it will return the line characterized by `w` and `b` that separates the training data with the largest margin between the positive and negative class.

Since the decision boundary is a line in the case of separable 2D data, you can find the best decision boundary (the one with the maximum margin between positive and negative samples) by looking at lines that separate the data and choosing the best one (the one with the maximum margin). The maximum margin separator depends only on the support vectors. So you can find the maximum margin separator via a brute force search over possible support vectors. In 2D there can only be either 2 or 3 support vectors. *THINK ABOUT THIS.*

In order to implement your training function, you will need to write some helper functions:

Implement: Write a function that compute the distance between a point and a hyperplane. In 2D, the hyperplane is a line, defined by `w` and `b`. Your code must work for hyperplanes in 2D. Do not worry about higher dimensions.

```
dist = distance_point_to_hyperplane(pt, w, b)
```

Implement: Write a function that takes a set of data, and a separator, and computes the margin.

```
margin = compute_margin(data, w, b)
```

Implement: You will need to write the following function to test new data given a decision boundary. The following function will output the class `y` for a given test point `x`.

```
y = svm_test_brute(w,b,x)
```

Use the above helper functions when writing your training function.

2 Multi-Class Classification (50 points)

Now, let's assume we're working with data that comes from more than two classes. You can still use an SVM to classify data into one of the Y classes by using multiple SVMs in a one-vs-all fashion. Instead of thinking about it as class 1 versus class 2 versus class 3... you think of it as class 1 versus not class 1, and class 2 versus not class 2. So you would have one binary classifier for each class to distinguish that class from the rest.

We will provide you with a function to generate training data:

```
[data, Y] = generate_training_data_multi(num)
```

Similarly to the binary data generation function, `num` indicates the particular set of data, and there are 2. Each dataset has a different number of classes. The classes are identified from 1 to `Y`, where `Y` is the number of classes.

We have also provided you with a function to plot the training data so you can visualize it:

```
plot_training_data_multi(data)
```

Implement: Implement the following function:

```
[W, B] = svm_train_multiclass(training_data)
```

This function should use your previously implemented `svm_train_brute` to train one binary classifier for each class. It will return the `Y` decision boundaries, one for each class (one-vs-rest). `W` is an array of `w`s and `B` is an array of `b`s where $w_i x + b_i$ is the decision boundary for the i^{th} class.

Implement: You will also implement a test function:

```
y = svm_test_multiclass(W,B,x)
```

This function will take the C decision boundaries as input and a test point x, and will return the predicted class, c. There are two special cases you will need to account for. You may find that a test point is in the "rest" for all of your one-vs-rest classifiers, and so it belongs to no class. You can just return -1 (null) when this happens. You may also find that a test point belongs to two classes. In this case you may choose the class for which the test point is the farthest from the decision boundary.