50.021 – AI

Alex

Week 01: Discriminative ML - quick intro

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

Due: week3 Thursday, 6nd of June, 6pm

This is a homework which is larger (twice as much) than the usual coding homeworks.
Goal: Run a small ML project from scratch. When I get a new dataset, I have to do these steps 5 times per year at least.
The goal is to let you do a bit recap on how to run a machine learning project in principle.

- Download from edimension the imageclef2011 image features, the set of all labels in `concepts_2011.txt` and the image labels `trainset_gt_annotations.txt`. The images in `photos_8000.zip` are not needed right away.
- Split the images into 60 percent per class for training, 10 percent per class for validation, 30 percent per class for final test. What is the difference to a random $60-10-30$ split of the whole data as compared to split class-wise? Why I asked you to split classwise ? Explain in at most 8 sentences.
- idea: use a linear SVM (e.g. scikit-learn) with python3 interface. If you use another kernel, please notify clearly in the report.
- for the first experiment take only the features from those images who have as label either spring or summer or winter or fall. that should be 1353 samples.

- That results in a multi-class dataset with mutually exclusive labels. Thus you can train 4 binary svms, one for each class in one-vs-all manner. Each svm is trained on the training dataset using all the training data.
- At test time you predict the class-label as the index of the svm with the highest prediction score.
- This method has one free parameter - the regularization constant. Find the best regularization constant from the set $0.01, 0.1, 0.1^{0.5}, 1, 10^{0.5}, 10, 100^{0.5}$ by repeatedly training on the training set and measuring performance on the validation set. Use as performance measure the class-wise accuracy averaged over all 4 classes.
- Once you have the best value for C, train on train + validation. then report performance of that classifier on the test set
- Submit your code (python3), including the code for splitting, and your saved train val test splits (.npy),
- Submit a short report showing the two performance measures on validation and testset:

  vanilla Accuracy

$$\frac{1}{n}\sum_{i=1}^{n} 1[f(x_i) == y_i]$$

  class-wise averaged accuracy:

$$\begin{aligned} A =& \frac{1}{C}\sum_{c=1}^{C} a_c \\ a_c =& \frac{1}{\sum_{i=1}^{n} 1[y_i == c]} \sum_{i=1}^{n} 1[y_i == c] 1[f(x_i) == c] \\ =& \frac{1}{\sum_{(x_i,y_i):y_i=c} 1} \sum_{(x_i,y_i):y_i=c} 1[f(x_i) == c] \end{aligned}$$

Useful things in python:

```
File handling:
os.path.basename(...)
os.path.join(...)
os.path.isdirectory(...)
os.makedirs()

for reading one line of the labels file trainset_gt_annotations.txt:
line.rstrip().split() returns you a list where the first entry is the filename,
and the following 99 entries are the labels as strings (not as integers!!).
```

```
they need still to be converted into integers

also helps to extract parts from a filename:
position=pythonstring.find(someotherstring)

loading of numpy arrays:
np.save(...)
np.load(...)

sklearn.svm.*
yourwhateversvm.predict(...) does not help you here,
you need the real valued scores,
not the label from one one-vs-all SVM
```