SGN-41007 Pattern Recognition and Machine Learning

Exercise Set 7: February 18–February 22, 2018

Exercises consist of both pen&paper and computer assignments. Pen&paper questions are solved at home before exercises, while computer assignments are solved during exercise hours. The computer assignments are marked by `python` and Pen&paper questions by `pen&paper`

1. `pen&paper` *Error rate confidence limits.*

   We train a classifier with a set of training examples, and test the accuracy of the resulting model with a set of $N = 100$ test samples. The classifier misclassifies $K = 5$ of those.

   a) Find the 90% confidence interval of the result. Hint: The classification accuracy can be modeled using binomial distribution, whose confidence intervals are discussed here:

      `https://en.wikipedia.org/wiki/Binomial_distribution#Confidence_intervals`

   b) Another classifier misclassifies only 3 test samples. Is it better than the first one with statistical significance at 90% confidence level?

2. `pen&paper` *Design a regularized LDA classifier.*

   Let's revisit the LDA design of Exercise set 4, but add a regularization term. The non-regularized LDA solution is given by as

   $$\mathbf{w} = (\mathbf{\Sigma}_0 + \mathbf{\Sigma}_1)^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$$

   The regularized solution with regularization parameter $\lambda > 0$ is defined as

   $$\mathbf{w} = (\mathbf{\Sigma}_0 + \mathbf{\Sigma}_1 + \lambda\mathbf{I})^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$$

   However, as the scale of $\mathbf{w}$ is not important—only the direction—let us use an alternative definition instead:

   $$\mathbf{w} = \lambda (\mathbf{\Sigma}_0 + \mathbf{\Sigma}_1 + \lambda\mathbf{I})^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0).$$

   This definition avoids the convergence of $\mathbf{w}$ towards zero as $\lambda \to \infty$.

   a) Compute the regularized LDA weight vector[1] for $\lambda = 100$ and

   $$\boldsymbol{\mu}_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad \boldsymbol{\mu}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$
   $$\mathbf{\Sigma}_0 = \begin{pmatrix} 3 & -2 \\ -2 & 2 \end{pmatrix} \qquad \mathbf{\Sigma}_1 = \begin{pmatrix} 3 & -2 \\ -2 & 2 \end{pmatrix}$$

   b) Where does $\mathbf{w}$ converge as $\lambda \to \infty$?

[1] Remember the inversion rule for 2×2 matrices:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

3. `python` Let us use a pretrained VGG16 model for last week's GTSRB experiment. Instead of using the custom ConvNet of last week, initialize a VGG16 net as described on Slide 15 of the slide set at `http://www.cs.tut.fi/courses/SGN-41007/slides/Lecture6b.pdf`. Add dense layers after the convolutional pipeline such that `model.summary()` reports the following (top of listing omitted):

```
block5_conv3 (Conv2D)         (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)    (None, 2, 2, 512)         0
_________________________________________________________________
flatten_1 (Flatten)           (None, 2048)              0
_________________________________________________________________
dense_3 (Dense)               (None, 100)               204900
_________________________________________________________________
dense_4 (Dense)               (None, 2)                 202
=================================================================
Total params: 14,919,790
Trainable params: 14,919,790
Non-trainable params: 0
```

Compile and run the net. Note that you will need a GPU (*e.g.*, TC303 machines) for training this net.

4. `python` Apply the recursive feature elimination approach (`sklearn.feature_selection.RFECV`) with logistic regression classifier for the arcene dataset. The data can be downloaded in `*.mat` format from:

   `http://www.cs.tut.fi/courses/SGN-41007/exercises/arcene.zip`

   Use `scipy.io.loadmat` to open the file. Note that your have to ravel `y_train` and `y_test` so that `sklearn` will accept them.

   a) Instantiate an RFECV selector (call it `rfe` from now on). To speed up computation, set `step = 50` in the constructor. Also set `verbose = 1` to see the progress.
   b) Fit the RFECV to `X_train` and `y_train`.
   c) Count the number of selected features from `rfe.support_`.
   d) Plot the errors for different number of features:
      `plt.plot(range(0,10001,50), rfe.grid_scores_)`
   e) Compute the accuracy on `X_test` and `y_test`. You can use `rfe` as any other classifier.

5. `python` Apply L_1 penalized Logistic Regression for feature selection with the arcene dataset. Find a good value for parameter C by 10-fold cross-validating the accuracy. Study the sparseness of the solution: how many features were selected?

   a) Instantiate a LogisticRegression classifier. Set `penalty = 'l1'` in the constructor.

b) Cross validate the accuracy of a range of `C` values (see earlier exercises).

c) Fit the LogisticRegression to `X_train` and `y_train`.

d) Count the number of selected features from `clf.coef_`, where clf is your logistic regression classifier.

e) Compute the accuracy on `X_test` and `y_test`.