Manuel Haussmann
mlhd1516@gmail.com

Exercise 4

Deadline: 08.06.2016

1 Multi-Class Classifiers

The goal of this exercise is to compare different multi-class classification methods. Mainly we will rely on combinations of binary classifiers. I suggest that you use a random forest but you can also try support vector machines. If not specified otherwise use a tree depth of 20 for the RF. While you create the code, keep in mind that we want to do a cross validation of all classifiers in exercise 1.6.

1.1 Dataset Preparration(5 Points)

As usual, load the digits dataset from sklearn

```
from sklearn.datasets import load_digits
digits = load_digits()
data = digits["data"]
images = digits["images"]
target = digits["target"]
target_names = digits["target_names"]
```

First, normalize the data and separate the samples by class. Therefore, we introduce the matrix X^k for every class k that contains all feature vectors X_i if $Y_i = k$. Normalize the feature matrix X by

- subtracting the mean vector $\bar{X}$ from every feature vector X_i
- dividing each feature vector by the standard deviation of the whole dataset
- separate the datasets by class (i.e. create X^k)

1.2 One vs Rest(8 Points)

Train 10 binary random forest classifiers. For each classifier use X^k as positive examples and all other X^l with $l \neq k$ as negative examples. Now, we have two possibilities for prediction. We can predict the class with the highest probability from the random forest (score of the SVM) or take each binary output directly. The second choice may lead to binary codes that can not be translated into a class label(i.e. all entries -1 or more than one $+1$). In our case we will count this *unknown* class as an error.

The uneven ratio of negative and positive samples may affect out classifier. To prevent this we can weight the error by the number of provided examples. In sklearn 0.16 or higher you can do this out of the box with with

```
sklearn.ensemble.RandomForestClassifier(class_weight="auto")
```

. Alternatively, you could generate a new dataset (subsample) with randomly drawn negative samples with

```
sklearn.ensemble.RandomForestClassifier(class_weight="subsample")
```

Task: Create 4 One vs Rest classifiers:

- with subsampling and without weighted loss
- with argmax or binary prediction

Which method profits from the weighted loss?

1.3 Error-Correcting Output Code(12 Points)

The previous binary encoding was not ideal since we did not use the full code information. To do this we define a binary code word of length $P \in \{9, 10, 11\}$ for every class. Good code words will have a large distance to each other and therefore allow for corrections if the predicted binary word does not exactly match. The theoretically maximal achievable distance is

$$d = P - Q + 1$$

where Q is the log of the number of classes (i.e. $Q = 4$).

First we want to derive codes from the eigenvectors of the correlation matrix. To this end,

- calculate normalized mean vector for each class

$$\vec{m}_k = \frac{\sum_i X_i^k}{||\sum_i X_i^k||}$$

- compute correlation matrix form class means

$$corr \in \mathbb{R}^{K \times K} \quad \text{with} \quad corr_{ij} = \vec{m}_i \vec{m}_j$$

- calculate the eigenvectors and eigenvalues of the correlation matrix
- generate a code of length 10 by converting the eigenvectors of each class to binary with an elementwise sign operation
- for codes of length 9 remove the vector with smallest eigenvalue
- for codes of length 11 add an arbitrary bit to each vector. Use the same amount of +1 and -1.

Notation: Let $C \in \{-1, 1\}^{K \times P}$ be the code word matrix, where every column is an eigenvector of $corr$. We choose every row C_k as the code word of class k.

Alternatively, draw a random $C \in \{-1, 1\}^{K \times P}$ and build the classifier from there.

Training: Train one binary random forests classifier for each of the P letters in the code word. Use all classes where the letter is '+1' as positive examples and '-1' as negative examples.

Task:

- Compute the maximum code distance

$$\frac{1}{2}(P - \max(C \cdot C^T - \mathbb{1} \cdot P))$$

  of the random and the eigenvector encoding and compare it to the theoretical maximum of
- Find a method for prediction, where you map all possible binary outputs to the class labels (hint: consider hamming distance).
- Create 6 ECOC classifiers with:
  - for $P \in \{9, 10, 11\}$
  - with random and eigenvector codewords

1.4 One-vs.-one(6 Points)

Method 1:

- train $\frac{1}{2}K(K-1)$ classifiers on every pair of classes
- For prediction: Apply all $\frac{1}{2}K(K-1)$ classifiers to the unseen sample. The class that gets the highest number of "+1"predictions gets predicted by the combined classifier

Method 2:
We can reduce the number of classifiers by greedily classifying in sequence. Therefore, we introduce $K-1$ binary classifiers G_k $\forall k \in \{0, ..., K-1\}$, where each G_k is trained with negative examples from class k and positive examples from class $k+1$. The following algorithm can be used for prediction of sample s:

Algorithm 1 One vs One | Method 2

```
a = 0
b = K
while a ≠ b do
    if p(s|G_a) > 1 − p(s|G_{b−1}) then
        b ← b − 1.
    else
        a ← a + 1.
return a (predict class a)
```

where $p(s|G_k)$ is the probabilistic prediction of classifier G_k for sample s.
Task: Create 2 classifiers: One for each method.

1.5 Multi-Class Random Forrest(4 Points)

For a comparison train one multi-class random forrest with 10 times as much trees as previous methods.

1.6 Compare(6 Points)

Compare the average Error of each classifier over 10-fold cross validation.

Regulations

Please hand in the python code, figures and explanations (describing clearly which belongs to which). Non-trivial sections of your code should be explained with short comments, and variables should have self-explanatory names. Plots should have informative axis labels, legends and captions. Please enclose all results into a single .pdf document and hand in the .py files that created the results. Please email the solutions to `mlhd1516@gmail.com` before the deadline. You may hand in the exercises in teams of maximally three people, which must be clearly named on the solution sheet (one email is sufficient). Discussions between different teams about the exercises are encouraged, but the code must not be copied verbatim (the same holds for any implementations which may be available on the WWW). Please respect particularly this rule, otherwise we cannot give you a passing grade.