Week 5 & 6: Image Classification
Create your own "Google Similar Images" system

Efstratios Gavves *

December 3, 2013

1 Summary

1.1 Provided:

- Handout
- Basic code and functionality
- Dataset

1.2 Requested:

- A report. **The questions which you need to answer in the report will be marked with red font in this handout.** The report should be delivered at `efstratios.gavves@gmail.com`. The **deadline** for delivering report 3 is Sunday 15th of December, 23:59:59. Delay in delivering the report will be penalized with **minus 1 point per day**.
- A zip file with the implemented code. The places where you are suggested to put your implemented code (this time in `week56.py`) will be marked with comments

  `[# WRITE YOUR CODE HERE], or something similar.`

  You **should not** write code where there is a comment

  `[# DO NOT TOUCH]`

  We do so to ensure that the final code will look as clean as possible. That way we will use our time in evaluating your code and giving you feedback, and **not debugging it** or trying to understand it. We are not responsible for debugging, please make sure the code you send us works. If there are bugs you will be asked to re-submit once **at most**.

*Email: `efstratios.gavves@gmail.com`, Room: C. 3.244, Office hours: Mon 17.00-18.00

1.3 Useful links:

- `https://www.enthought.com/` Canopy Enthought environment
- `http://effbot.org/imagingbook/pil-index.htm` Python Imaging Library Handbook
- `http://docs.scipy.org/doc/` Python Numpy and Scipy Library documentations
- `https://piazza.com/uva.nl/fall2013/uva/home` Piazza.com, the electronic embodiment of our class, where you can ask questions and we will answer. Asking questions here is good, as everybody can profit from the answer.

2 Preliminaries

2.1 Summary

In the final assignment we will get more familiar with classifiers, that is functions which predict the label of an image. We will first work on k-NN classifier, which resembles significantly what we did last week with the query-by-example retrieval. We will see how one can tune the crucial parameter k for k-NN classification via cross-validation, so that maximum performance is ensured. Next, we will briefly discuss the support vector machine classifiers, mainly focusing on the intuition behind them. Finally, we will test the support vector machine classifiers in real image examples.

The learning objectives are:

- Understand k-NN classifier
- Understand cross-validation
- Understand the intuition behind support vector machine classifiers
- Evaluate and compare the k-NN and the support vector machine classifiers in real data.

2.2 Download the provided package for Week 5 & 6

First download the assignment package from:

`http://staff.science.uva.nl/~tmensink/webdata/multi_week5_6.zip`.

Note: different location than before!

Extract the package in your working directory $main_dir, like in the previous week e.g.,

`C:\Users\username\multimedia_course`

Before proceeding please **make a backup** of your previous code. Taking a backup is essential, better be safe than sorry.

3 Introduction to Classifiers

In this week we will focus on building our own image classification system. The goal is to classify images into predefined classes, for example *car* or *non-car*. Intuitively, you could think of a classifier as a line, which separates the items from each other according to some rules that we need to discover.

To learn the classifiers, we make use of a *labelled dataset*, that are pairs consisting of an image and a label, such as (im1.jpg, car), (im2.jpg, car), ..., (im243.jpg, elephant), etc. We use the knowledge represented by this dataset to optimize a classification function. Their output is a classification line (for 2 dimensional data), or a hyperplane (for higher dimensional data), that is supposed to divide the data, see Fig. 1. In our case the images are represented by the bag-of-words histograms, and therefore we learn a classifier in the bag-of-words histogram *space*.

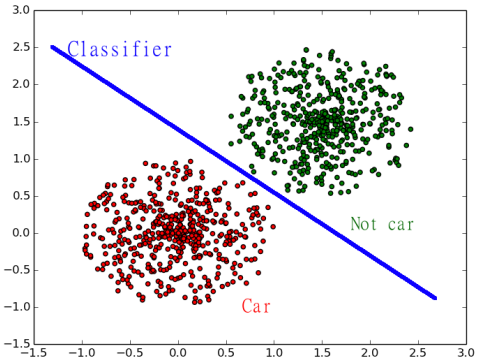

Figure 1: How does a classifier look like?.

In practice, we split our dataset into a training partition and a testing partition. We then use the training partition to train our models and fit the various parameters and the testing partitions to see how well our model works.

These two partitions should be strictly separated, that is no images from the training partition should be present in the testing partition. The reason is that if we train on the same data that we test our models, we cannot know whether in the end our model is accurate in describing new images. It could be that we just found a very particular way of explaining our images of the training data, without being able to generalize on unseen images. When the latter happens, we observe the so called "overfitting".

4 k-Nearest Neighbor (*k*-NN) Classifier

4.1 Description

The k-Nearest Neighbour classifier is one of the classic classification algorithms, it is based on a simple model, but it has some strong mathematical guarantees and advantages as discussed in last week lecture. The idea is simple, and quite close to what we did last week.

- Given a query image
- Rank all **training images** based on their visual similarity
- Use the **labels** from these train images
- Find the most frequent label among the top k most similar images
- Assign that as label for the query image

There are two important choices to make for k-NN classification: (1) which distance to use for the ranking of images, and (2) the value of k. The first we have discussed (in length) last week for image retrieval. Here we will implement a k-NN classifier and cross-validate the value of k.

First, load the dataset information using the following function

```
files, labels, label_names, unique_labels, trainset, testset =
week56.get_objects_filedata()
```

The output arguments are as follows:

- files: the filenames of the images
- labels: the labels of the images as integers, e.g. 1, 2, ...
- label_names: the labels of the images as words "car", "train", etc.
- unique_labels: the set of all possible label names
- trainset: the training set partition
- testset: the testing set partition

Load the Bag-of-Words histograms from the directory `../../data/bow_objects/codebook_100` for all the files. You can use the same function as you used in the previous week. Now, you should have the bag-of-words represenations for all images together with their labels.

Q1: Implement the k-NN classifier, by using the code that you used in the assignment of Week 3 & 4 for ranking images according to their *histogram intersection similarity*. Since we use histogram intersection similarity, you should l1 normalize the data first. After ranking the top k **training** images according to their similarity to the test image, use their labels to predict the label for the test image. Run the k-NN classifier for the test images `goal/15.jpg`, `bicycle/37.jpg`, `beach/31.jpg`, `mountain/37.jpg`. Paste the most similar images to their report and predict their labels using $k = 9$ nearest neighbors. Report the predicted labels.

Q2: In the standard k-NN classifier, the most common class-label among the top k images is returned as label for the query image. Could you come up with a different strategy to obtain a label for the k-NN classifier? Describe your strategy, implement it, evaluate the same images according your new strategy, and compare the results to the default strategy.

5 Deciding which k via Cross Validation

A crucial parameter for ensuring good performance for the k-NN classifier is the number of k images that we consider for making our prediction. Although one can select a number at random, it is always better to make an educated guess that is actually optimal for certain criteria. In our case the criteria is the classification accuracy. For one class, e.g "car" the classification accuracy for the test images is equal to

$$acc = \frac{\#true\ positives}{\#true\ positives + \#false\ negatives}, \tag{1}$$

where "#true positives" is the number of car images, which were correctly classified as car images and "#false negatives" is the number of car images that were not predicted correctly. Apparently, the sum "#true positives + #false negatives" is equal to the total number of car images in the test set. The mean classification accuracy is the average of the classification accuracies for all our classes in the dataset.

Q3: Implement the accuracy evaluation measure, and evaluate it for the k-NN classifier (using $k = 9$) on the images from the test set. Report the class accuracy **per class** and the mean. The mean should be (close to): 0.31

In this part, we try to find a good value for the parameter k. Obviously, we cannot use the testing images to evaluate the classification accuracies. Therefore we can use only the training data, and we perform cross-validation of the parameter k, based on mean classification accuracy.

The concept of cross validation is a simple one, which we will briefly describe next by an example. First, we split our training set into smaller subsets, let's say 3 subsets of equal size. Then, we run several rounds of experiments, where we keep the combination of two (randomly chosen) out of the three subsets as our "actual' training set. Then we measure the mean classification on the third one, which we call the "validation set". For this setting of "actual" training and "validation" sets we perform k-NN classification for different values of k (or any other parameter we could have), e.g. k=1, 3, 5, 9. We repeat the same process for multiple rounds, where in each round we pick two different subsets as our "actual" training data, until all combinations of subsets are visited. In the end we will have a mean classification accuracy measure per round and per value of k. We finally pick the value of k for which the performance was the most accurate on average for all combinations of actual training and validations sets.

Q4: Perform cross-validation on the data, experimenting for the values of k={1, 3, 5, 7, 9, 15}. Report the mean class accuracy for each value of k, and evaluate the optimal value of k on the test set. Provide for the optimal k value the **per class** accuracies for the test set (are they the same as during cross-validation?) and describe your observations.

6 Support Vector Machine (SVM) Classifier

6.1 Introduction to SVMs

The support vector machines are one of the most popular, off-the-shelf, classifiers. The reason behind their popularity is their high accuracy, their robustness against outliers and their overall good generalization properties.

Intuitively, support vector machines are designed to compute the classification line which lies in the middle between the example images of the classes that are being confused the most. It is much easier to illustrate this by a figure, for example Fig. 2.

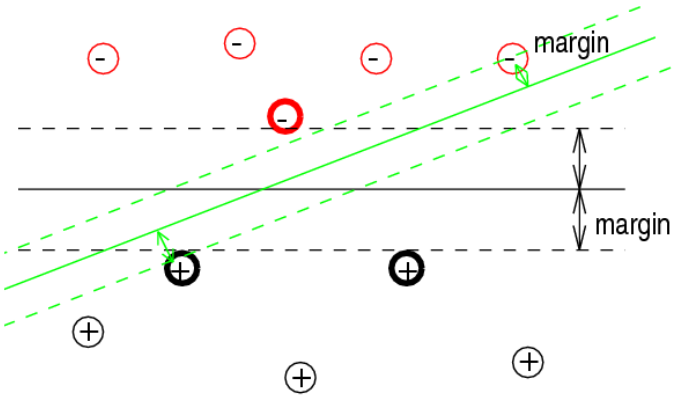

Figure 2: Support vector machines maximize the margin (white space) between two classes. The bold circled points are the so called "support vectors", which practically define the margin.

In the figure we can see that this line that lies in the middle guarantees the maximum of empty space between the two classes, or alternatively a maximum margin that separates the areas of the two classes. The notion of margin is a crucial one in the support vector machine field, due to which support vector machines are also often referred to as *max-margin* classifiers. Formally, the hyperplane with max-margin is computed by minimizing the following function:

$$J(w,b) = C \sum_i \max\left\{0, 1 - y_i(w^T x_i + b)\right\} + \Omega(w), \tag{2}$$

where x_i is the bag-of-words histogram of image i, $y_i = \{-1, 1\}$ is the label of image i (e.g. 1 for car, and -1 for not-car). The w and b are essentially the output of the SVM, w being a weight vector that is multiplied with the bag-of-words histogram x_i and b is a single number, the intercept or bias.

To classify an image we use the w and b and compute:

$$\hat{y} = \text{sign}\left(w^T x_i + b\right). \tag{3}$$

We have a 2-class classification problem, where we say that an image is either a "car" or a "not-car", hence the label y_i can take two values. For mathematical convenience we say that these values are "+1" for the positive class, e.g. "car", and "-1" for the negative one, e.g. "not-car".

The eq. (2) is the error function, which tells us when our line w is a bad one for a given image pair (x_i, y_i). If we assume that we have a good classifier, then the error function should return very small values, when our classifier predicts correctly the label of an image. On the other hand when our classifier makes a wrong prediction, the error function should return larger values. In fact, there might be cases when our classifier is **very** confident that an image shows a

"car", where in fact it shows an "elephant", which is a big mistake. Hence, it might be a good idea that the bigger the mistake, the larger the error values that our error function should return.

We will now manually test for simple toy data four classifiers and you will need to tell which one of those is the correct support vector machine classifier for these data. That is you need to tell us which of the classifiers guarantees a maximum margin. We will use the SVM function provided by the `scikit` library in Python. This library is normally available after paid subscription in Python, however, if you sign up to Canopy with your UvA email, then you will be able to download it for free. First, generate the toy data according to

```
data, labels = week56.generate_toy_data()
```

Then, use the function

```
svm_w, svm_b = week56.generate_toy_potential_classifiers(data, labels)
```

This function generates 4 classifiers, stored as rows for the two output arguments `svm_w` and `svm_b`.

Q5: For each of the 4 classifiers, classify the dataset, and visualize both the true label as well as the predicted label for each point. Which one of the four classifiers is the support vector machine, that is which one of the four classifiers minimizes the max-margin error? Motivate your choice.

Q6: Use the `scikit` library to train a SVM classifier on the provided data and labels:

```
 svc = svm.SVC ...
pred = ...
```

using a linear kernel, and $C = 1$. Illustrate the resulting classifier and report your results.

Note that in order to get the weight vector for the SVM (which you could use for visualization), you need to call `svc.coef_`, whereas to get the intercept one needs to call `svc.intercept_`. Here, the "svc" stands for the output of the SVC scikit function.

6.2 Non-linear classification using linear SVMs

Linear classifiers assume that the data is (almost) linear separable in the chosen representation of the data set. Or in other words, we can find a hyperplane in the bag-of-words space to separate car vs non-car. However, this is a rather strong assumption, which is not always the case.

We start with data from two classes, illustrated in Fig. 3, which is generate using the function:

```
data, labels = week56.generate_ring_data()
```

As we can see, the red class groups all the points that fall inside a ring around (0, 0) with radius range $[0, 1]$, whereas the second class groups all points inside a ring around (0, 0) and a radius range $(1, 2]$.

Q7: Run your linear SVM classifier on your ring data and display and evaluate your results. Explain visually whether the linear classifier is a good choice for

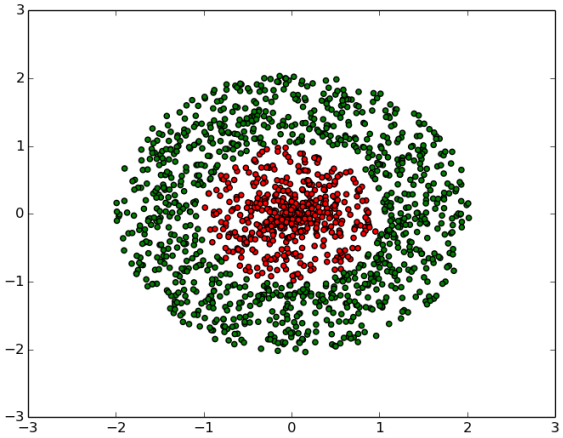

Figure 3: Ring data. The red points are the first class, the green points are the second class.

our data. Furthermore, report your classification accuracy for the two classes. Can you think of another type of a classifier for this specific task? Draw your preferred classification line into the figure

A possible approach for non-linear classification using linear classifiers is by using *feature transformations*:

$$x' = \phi(x) \tag{4}$$

where $\phi(\cdot)$ is a non-linear function, e.g. $x' = \sqrt{x}$. In words: we transform our input features. In this new space x' we learn a linear classifiers, which result in a non-linear classification in the original space x.

Q8: Write down two other different non-linear transformations of the data, and discuss whether they would help for the ring data. *If you are smart enough to come up with the transformation described below, you should include a third non-linear transformation.*

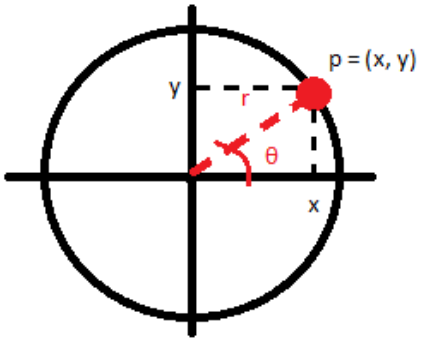

Figure 4: Polar coordinates.

So far the theory, lets get back to the ring data. We know that our data have a very strong circular-like pattern, being distributed around a center within some radius range. Let us focus on a specific point $p_i = (x_i, y_i)$, which we visualize in Fig. 4 with a red dot. As we can see in the figure, this point lies exactly on a circle with radius r_i and angle θ_i. For example, the point $p_0 = (x_0 = 1, y_0 = 1)$ lies on a circle with radius $r_0 = 1$ and angle $\theta_0 = \pi/4$ ($\pi/4 = 45°$). Similarly, another point p_j lies on another circle with radius r_j and angle θ_j. In a similar fashion we can repeat for all points and find on which circles they lie on, by calculating the respective radii and angles. As a result, in the end we can replace each point $p = (x, y)$ with the equivalent $p = (r, \theta)$. These are called polar coordinates.

For a point $p_i = (x_i, y_i)$ we can find the radius and the circle on which the point lies using the following two equations:

$$r_i = \sqrt{x_i^2 + y_i^2} \tag{5}$$

and

$$\theta_i = \arctan2(y_i, x_i) \tag{6}$$

We will use the polar-coordinates to obtain $p_i' = (r_i, \theta_i)$.

Q9: Transform all your points you acquired to their polar coordinates (r, θ). For the angle θ you can use the python function for calculating the inverse tangent (`numpy.arctan2(y,x)`). After having compute the polar coordinates, visualize the points using their polar coordinates.

Q10: After having computed the polar coordinates for the points, run again the linear SVM classifier. Visualize the classifier output in both (e.g. using the predicted labels for all data points) *both* in the polar coordinates and in the original space. What do you observe that happens after you use the polar and not the linear coordinates? What is the classification accuracy for your new polar representations?

6.3 Classify images

By now we have described the use of linear SVM classifiers on simple toy data. We will proceed with a real application, where we will classify complicated images instead of simple two dimensional points. Similar to the assignment of the

previous week, we provide you the Bag-of-Words representations of the images of different classes, such as *airplane, train, etc.*. You can find the histograms stored in the `data/bow_objects`, and you can load them with the same functions you used in the previous week.

The `scikit` library also allows to classify into more than 2 classes. In that case the labels should be the integer representing the class, $y_i = \{1, \ldots, 10\}$, e.g. $y_i = 1$ for airplanes and $y_i = 2$ for trains. How this is handled exactly could be a bonus question (see below), but is not of utmost important.

Q11: Load the histograms for all images and all classes. Run the linear SVM classifier for them using different codebook sizes. Tune with cross-validation on the training data the SVM parameter C (use 3 splits of the train set). Classify all your test images for all your classes and report the mean classification accuracy. What do you observe? Repeat for different codebook sizes and report which codebook works the best for you.

An important aspect when it comes to evaluating image classifiers is the qualitative check-up, that is where your image classifier works and more importantly, where it fails.

Q12: Go over the results of your best-tuned SVM classifiers. Which classes are best recognized, could you visualize this? Can you explain why? Which class are the least recognized? Can you explain why?

6.3.1 Comparison of k-NN vs SVM

Q13: Now, use the tuned k-NN classifier to predict the labels for all the testing image for all classes. Report the mean classification accuracy and compare the results with the one obtained by the support vector machines. Repeat for different codebook sizes and report your observations.

7 Bonus

If you are interested in the bonus question, please contact us and we will explain one of the the three possible bonus questions:

1. The Kernel Trick
2. Learning the SVM weight vector
3. Multi-class image classification