ME41105 - IV Assignment 1
Visual object detection

Dariu Gavrila, Julian Kooij
Ewoud Pool, András Pálffy, Joris Domhof
Intelligent Vehicles group
Delft University of Technology

September 27, 2017

About the assignment

Make the assignments in student pairs, you receive both one grade. Please read through this whole document first such that you have an overview of what you need to do.

This assignment contains *Questions* and *Exercises*. You should address all of the questions in a 1 or 2 page report (excluding plots and figures). *Please provide separate answers for each Question in your report, using the same Question number as in this document.* Your answer should address all the issues raised in the Question, but typically should not be longer than a few lines. The Exercises are tasks for you to do, typically implementing a function or performing an experiment. Do not directly address the exercises in your report. Instead, you should submit your solution code together with the report. Experimental results may be requested in accompanying questions.

You will be graded on:

1. Quality of your answers in the report: Did you answer the Questions correctly, and demonstrate understanding of the issue at hand? All Questions are weighted equally.
2. Quality of your code: Does the code work as required?
3. Quality of presentation: Is your report readable (sentences easy to understand, no grammar mistakes, clear figures)? Is the code you wrote clear and commented?

Submitting

Before you start, go to the course's Brightspace page, and enroll with your partner in a lab group (found under the 'Collaboration' page). To submit, upload two items on the lab assignment page (found under the 'Assignments' page):

pdf attachment	A *pdf* with your report. Do not forget to add your student names and ids on the report.
zip attachment	A *zip* archive with your Matlab code for this assignment Do NOT add the data files! They are large and we have them already ...
deadline	**Friday, October 20 2017, 23:59**

Note that only a single submission is send for your group, and only your last group submission is kept by Brightspace. The due deadline is enforced by Brightspace, so submit on time!

Getting assistance

The primary occasion to obtain help with this assignment is during the lab practicum contact hours of the Intelligent Vehicles course. An instructor and student assistants will be present at the practicum to give you feedback and support.

If on the other hand you find errors, ambiguously phrased exercises, or have another remark about this lab assignment, please contact Julian Kooij (J.F.P.Kooij@tudelft.nl).

Remember that for help on what a specific Matlab command `somefunction` does or how to use it, use can type from the Matlab command line `help somefunction`, or `doc somefunction`.

Visual object detection

In this lab assignment we will study and evaluate feature extraction and pattern classification algorithms that can be used for video-based pedestrian recognition.

Our first goal is to investigate which classifiers give the best result in distinguishing rectangular region proposals as belonging either to the 'pedestrian' or 'non-pedestrian' class. We have a dataset containing 3000 samples (1500 pedestrian and 1500 non-pedestrian) for training. For testing, 1000 samples (500 pedestrian and 500 non-pedestrian) are provided, see Figure 1.

The pedestrian and non-pedestrian samples are provided in terms of three feature sets:

- `data_hog.mat`: Histograms of oriented Gradients (HOG) features
- `data_lrf.mat`: Local Receptive Field (LRF) features
- `data_intensity_25x50.mat`: Gray-level pixel intensity

1 Eigen-pedestrians

Exercises refer to code sections in `assignment_eigen_pedestrians.m`.

Let us start by exploring the dataset a bit. It may be difficult to interpret the LRF and HOG feature representations, but it is possible to visualize the gray-level intensity images by resizing them to their original size. Note that these 25×50 pixel images have been reshaped to 1×1250-dimensional vectors. To restore one vector to its original size, the Matlab command `reshape` can be used. After reshaping the matrix, it can be visualized by using `imshow`. *Note:* `imshow` will automatically scale the intensities if you pass it an empty array as second argument, e.g. `imshow(I, [])`.

> *Exercise 1.1.* Visualize several pedestrian and background samples from the training data. **Note:** You do not need to start from scratch, a lot of the boilerplate is already given in the provided assignment script referred to in the boxed note at the top of this section. In the script, look for code block with the comment 'Exercise 1.1'. You will see that you only

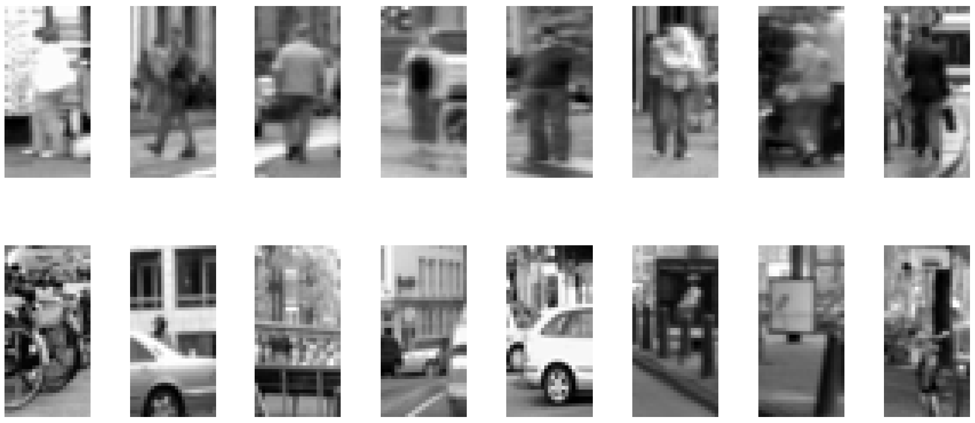
Figure 1: Examples of pedestrian and non-pedestrian class samples in the training data.

need to complete the function `imshow_intensity_features.m`. For instance, this first exercise can be solved using the correct calls to `reshape` and `imshow` only.

Some classifiers may not be able to handle the high dimensionality of the input data. Principal Component Analysis (PCA) is one method to reduce the data dimensionality by projecting it into a linear subspace that maintains most of the variance in the data, see Appendix A.

Question 1.1. For each of the three feature types, what is the maximum number of PCA components? Motivate your answer.

In this section, we shall study using PCA on the gray-level intensity features. For PCA you can use the built-in Matlab function `pca` (or `princomp` on older Matlab versions). Notice that to correctly project data onto the PCA dimensions, the mean vector should be subtracted from the data, so this vector needs to be computed too.

Exercise 1.2. Compute the principal components from the dataset, and visualize the mean, and the first 10 principal components as images. **Note:** Be aware that there are different conventions to represent feature vectors. For instance, in math notation (as in Appendix A), features vectors are typically expressed as column vectors. But in the code and data matrices, the features are given as rows. This difference matters when doing dot products between matrices and vectors.

These principal components are the eigen-vectors of the covariance computed on the given image intensity features. Since the image dataset contains pedestrians, its principal components can also be called *eigen-pedestrians*.

Question 1.2. Include images of the “mean” pedestrian, and the 10 eigen-pedestrians in your report. How do you interpret the light/dark regions in the eigen-pedestrians? What color would a PCA weight of ”0” have in these images? And, what color would an intensity of ”0” have in the intensity images from Exercise 1.1?

Exercise 1.3. Project intensity data of both the pedestrian and background training samples to the first three PCA components. After this projection, each sample will be represented by a 3D vector in the PCA space. Create a 3D point cloud of the 3D vectors of both classes.

Question 1.3. In the 3D plot, which axis has the largest amount of variance? Is this axis alone sufficient to separate the two classes in our dataset? Motivate your answer.

Exercise 1.4. For intensity features, take the top-n PCA dimensions and project some 6 images from the training data onto the corresponding linear subspace, and then project the images back to the original image space, and display them. Do this for $n = 10$ and $n = 100$.

Question 1.4. Compare the original images to those obtained after projecting to and from the $n = 10$ and also to the $n = 100$ subspace. How does n affect the image quality? How much (in percentage) do the PCA projections reduce the feature size compared to the original intensity image feature?

Exercise 1.5. Now for all three feature types, make plots of the percentage of explained variance (y-axis) vs. the number of PCA components/dimensions (x-axis).

Question 1.5. For each feature set, how many PCA dimensions should we keep to maintain 90% of the variance in the data? Include the plots the motivate your answer.

2 Pedestrian classification

Exercises refer to code sections in `assignment_pedestrian_classification.m`.

For each of the provided feature sets (HOG, LRF, and Intensity) we want to train the following two classifiers:

1. Linear Support Vector Machine (SVM) classifier (see Appendix B)
2. Gaussian-Mixture-Model (GMM) with Bayesian decision model (see Appendix C)

But *before* we train the classifiers, we have to decide if applying PCA dimensionality reduction is necessary. To decide on the number of dimensions to use, take into account how many free parameters each of the classifiers has, which need to be adapted during training (take into account input parameters, their dimensionality and their properties).

Question 2.1. How many free model parameters does a trained Linear SVM classifier have for M-dimensional feature vectors? Give a formula as a function of M, and motivate your answer. (Note that the model parameters do *not* include the parameters of the *training procedure*, such as number of data samples, number of iterations, or C which is used later in the assignment).

Question 2.2. How many free model parameters does a Gaussian-Mixture-Model classifier with K mixture components have for M-dimensional feature vectors? Give a formula as a function of M and K, and motivate your answer.

A good rule of thumbs is that to obtain meaningful results, there should be (much) less free parameters than training samples.

Question 2.3. For which of these classifiers is PCA dimensionality reduction necessary on this dataset? Motivate your answer.

Now we can train each classifiers on each of the feature sets, applying PCA first where appropriate. Afterwards, we compute the classification error (percentage of misclassified samples) for all trained classifiers on the test samples.

Exercise 2.1. Implement the SVM classifier by completing `train_SVM` for training, and `evaluate_SVM` for testing. See the comments in the code on available functions to train a SVM (don't worry about C, use $C = 2$). Train and test the SVM on all three feature sets.

Exercise 2.2. Implement the GMM classifier and evaluation in `train_GMM` and `evaluate_GMM`. As you will see, `train_GMM` provides already boilerplate code. Use the methods of Matlab's built-in `gmdistribution` object to fit GMM distributions on the data of each class, and to evaluate their pdf's on test data. Train and test GMM classifiers on all three feature sets, using $K = 5$ mixture components per class.

Question 2.4. What are the six classification errors that you obtained? Which feature/-classifier combination is best?

We could also evaluate the pedestrian classifiers using ROC curves (on y-axis: true positive rate [0,1], on x-axis: false positive rate), instead of the classification error measure. This requires logging the decision values (classifier outputs) on the test dataset.

Exercise 2.3. Complete the code for plotting the ROC curves. Look at Appendix D and Figure 3 for more information on the ROC plots.

Question 2.5. Which feature/classifier combination performs best? Include the ROC plots in your report to support your answer.

Now that we have these simple classifiers, we will take a look at techniques to further improve the performance.

Parameter selection and overfitting Both the SVM and the GMM have parameters that we can set before training the model, and which affect the final performance. In case of the SVM, this is the training parameter $C \in \mathbb{R}$, and for the GMM we have K, the number of mixture components (Actually, if we apply PCA, the number of PCA dimensions is a parameter too).

Till now we have kept these parameters fixed, but here we will experiment with optimizing them. When we train on some training data, we should always validate performance on *separate* testing data. Selecting parameters by minimizing the error on the training data is not a good indication of performance on new data, you will see that this leads to *overfitting*.

Exercise 2.4. Run the code block that evaluates the SVM for various values of C on both the training and testing data, and generate plots of the error as a function of C.

Question 2.6. If we evaluate on the *training data*, what is the lowest error that we can obtain, and for which C? What is the optimal parameter and error if we evaluate on the *test data*? Setting C too large leads to overfitting. But *why* does a large C decrease the training error, but increase the test error?

Exercise 2.5. Now implement your own code block to evaluate the effect of changing K in the GMM classifier on the HOG features. Try out these values for K ranging from 1 up to 7, and create again plots of the error as a function of K for evaluating on the training and test data. You can copy and alter the provided code from the previous exercise.

Question 2.7. If we evaluate on the *training data*, what is the lowest error that we can obtain, and for which K? What is the optimal parameter and error if we evaluate on the *test data*? Include the error plots in your report. Why does overfitting occur as we increase K?

Note: For the next exercise, you can keep using $C = 2$ for the SVM, and $K = 5$ for the GMM.

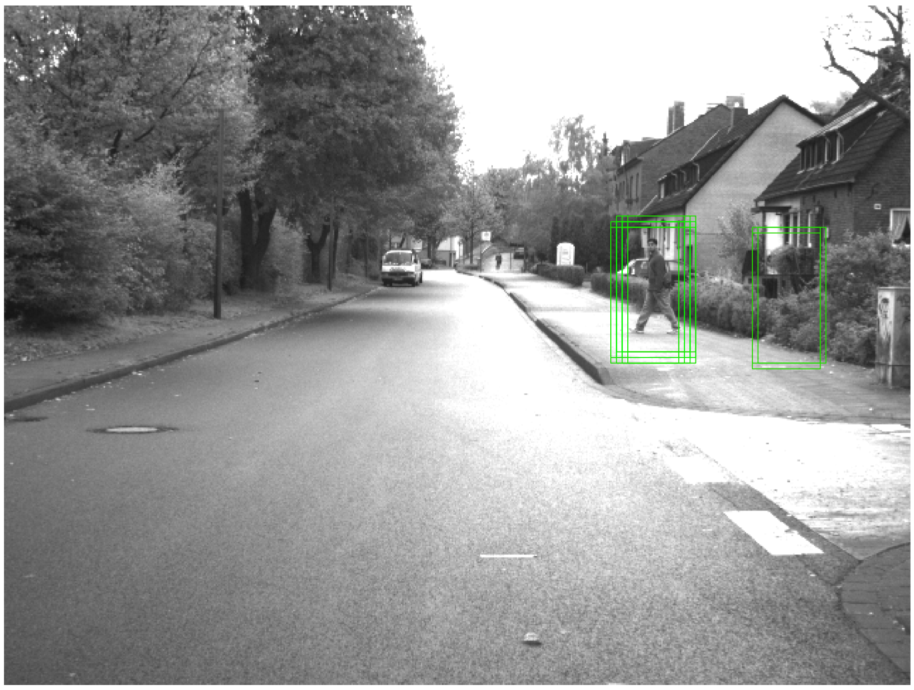

Figure 2: Pedestrian detection by classifying many region proposals. In this example, the green rectangles correspond to regions classified as pedestrians. Our aim is to avoid false positives and false negatives, though in practice misclassification errors may occur.

Multi-feature classification Multiple classifiers trained on distinct feature sets might provide complementary results. Intuitively, we should be able to benefit from having multiple distinct 'expert opinions'. Here we consider two approaches to fuse the decision values (classifier output) of the trained SVM classifiers on the HOG and LRF features, namely

1. fused output is the mean of the outputs of HOG/SVM and LRF/SVM
2. fused output is the maximum of the outputs of HOG/SVM and LRF/SVM.

Exercise 2.6. Evaluate both fusion approaches using ROC curves.

Question 2.8. Which fusion approach performs best? List the respective ROCs and discuss the effects you observe in your report.

3 Pedestrian detection

Exercises refer to code sections in `assignment_pedestrian_detection.m`.

Up to now, we have looked at pedestrian *classification*, i.e. deciding for a provided region proposal if it belongs to the pedestrian or non-pedestrian class. In this last assignment, we will move to pedestrian *detection*, which considers a broader question: In a given a target image, where are pedestrians located? A fairly simple but effective method to answer this question is to just divide the image into a large set of candidate region proposals of the right size and shape, and classify each of these regions using a trained pedestrian classifier. See Figure 2 for an example.

You are now provided with a new dataset with pedestrian and non-pedestrian samples, and computed HOG features, and also with a test video sequence of a pedestrian filmed from a moving vehicle. Furthermore, the region proposals have been pre-computed, and HOG features for each region are provided per video frame.

Exercise 3.1. Train and evaluate a linear SVM on the HOG features as good as you can.

Question 3.1. Explain your approach: Why did you (not) need to use dimensionality reduction? What value do you use for C and why?

Exercise 3.2. Train and evaluate a GMM on the HOG features as good as you can.

Question 3.2. Explain your approach: Why did you (not) need to use dimensionality reduction? What value do you use for K and why?

Question 3.3. Which of your classifiers is the best to use on this data? Include ROC plots to support your decision.

Exercise 3.3. Now apply your selected classifier on the region proposals of the video sequence, and visualize the regions which are considered 'pedestrian'.

Question 3.4. Studying the pedestrian detection results qualitatively (so by looking at the results, as opposed to quantitative evaluation using error statistics and ROC curves). What kind of false positives and false negatives do you observe? What would you suggest to counter these errors, and improve the results?

Acknowledgements

We thank Markus Enzweiler for his help in creating part of the data and early versions of the exercises used in this lab.

Appendix

A Principal Component Analysis (PCA)

PCA is a technique for reducing the dimensionality of the feature space. By analyzing how the data is distributed from training samples, it computes a linear subspace that maintains most of the variance of the input data. New data samples can later also be projected to this subspace.

From many data samples in an M-dimensional feature space, we can create a D-dimensional PCA subspace (where $D \leq M$) defined by $\mathbf{W}$ and $\mathbf{m}$. $\mathbf{W}$ is an $M \times D$ transformation matrix, and $\mathbf{m}$ the M-dimensional mean vector of the data. The columns of $\mathbf{W}$ are the first D eigen-vectors of the data covariance, and are also called the *principal components*. Note that each principal component is an M-dimensional vector too.

Each component also has an associated eigen-value. The components are ordered such that the i-th component is the eigen-vector with the i-th largest eigen-value λ_i. Even more, eigen-value λ_i is proportional to the amount of variance the data has along PCA subspace dimension i. The fraction of variance kept by principal component i is therefore $\lambda_i / \sum_{j=1}^{M} \lambda_j$.

Once the subpace is computed, we can apply the transformation to reduce the dimensionality of any M-dimensional data vector $\mathbf{x} \in \mathbb{R}^M$ to its reduced D-dimensional 'PCA' representation $\mathbf{x}^\star \in \mathbb{R}^D$ using the following linear equation (assuming all vectors are column vectors),

$$\mathbf{x}^\star = \mathbf{W}^\top (\mathbf{x} - \mathbf{m}). \tag{1}$$

The inverse back-projection can also easily be achieved,

$$\mathbf{x}' = \mathbf{W}\mathbf{x}^\star + \mathbf{m} \tag{2}$$

such that $\mathbf{x}'$ is the (approximate) reconstruction in the original M-dimensional feature space. If we keep all dimensions in the projection, such that $D = M$, then $\mathbf{W} \times \mathbf{W}^\top = I$. In other words, $\mathbf{W}$ is an orthonormal projection and therefore its transpose is its inverse $\mathbf{W}^\top = \mathbf{W}^{-1}$. However, typically we use $D \ll M$ so back-projection does not restore the original feature space exactly.

B Linear Support Vector Machine (SVM) classifier

Support Vector Machines are an advanced topic in Machine Learning. In this session, we will stick to Linear SVMs on two-class data, which form the basis of more complicated approaches.

A Linear SVM learns a hyperplane in the feature space that separates the data points of each class with maximal margin. The model parameters of the hyperplane are the M-dimensional weight vector $\mathbf{w}$ and a bias b which define the normal and offset of the plane. To test a new M-dimensional feature vector $\mathbf{x}$, the Linear SVM computes the following decision value:

$$d(\mathbf{x}) = \mathbf{w}^\top \cdot \mathbf{x} + b \tag{3}$$

The sign of this decision value determines the assigned class label, i.e. either it is assigned to the positive (i.e. pedestrian) class, $d(\mathbf{x}) \geq 0$, or to the negative (non-pedestrian) class, $d(\mathbf{x}) < 0$.

For learning Linear SVM parameters, the built-in `fitcsvm` Matlab function can be used. In case that you have an older Matlab version, you can also use the provided `primal_svm.m` function, which should let you obtain similar conclusions in the lab assignments.

Note that there is a *training* parameter $C \in \mathbb{R}$ (called *BoxConstraint* for `fitsvm`) which influences how the optimizer computes the linear decision boundary: For large C, the optimizer tries very hard to separate both classes, which means that its boundary is sensitive to outliers. For low C it results in a *soft margin* where a few samples may lie close to or on the wrong side of the boundary, if this enables a wider margin to separate most data points of both classes. While C is used during training, it is not part of the learned model in Equation (3).

C Gaussian-Mixture-Model (GMM) classifier

The Gaussian-Mixture-Model classifier belongs to a family of classifiers that model how the data from each class is distributed. In the training phase, these class conditional distributions are fitted on the available training data.

For a new test sample $\mathbf{x}$, the likelihood $P(\mathbf{x}|c)$ of the sample belonging to class c is evaluated for each class. Then, Bayes' rule can be applied to combine these likelihoods with the class priors $P(c)$ (i.e. how likely is each class *before* observing the feature) to obtained the class *posterior* distribution $P(c|\mathbf{x})$ (i.e. how likely is each class *after* observing the feature),

$$P(c|\mathbf{x}) = \frac{P(\mathbf{x}|c)P(c)}{\sum_c P(\mathbf{x}|c)P(c)}. \qquad \text{(Bayes' rule)} \qquad (4)$$

We then assign to $\mathbf{x}$ the class label with highest posterior probability, which is also called the *maximum a-posteriori* solution. In our two-class 'pedestrian' vs 'non-pedestrian' problem, we classify $\mathbf{x}$ as pedestrian if $P(c = \text{pedestrian}|\mathbf{x}) \geq \frac{1}{2}$. In the GMM the decision value is therefore $d(\mathbf{x}) = P(c = \text{pedestrian}|\mathbf{x})$, with the decision threshold $\frac{1}{2}$.

In case of the GMM classifier, the distributions $P(\mathbf{x}|c)$ of each class are modeled by a weighted mixture of K Multivariate Normal (i.e. 'Gaussian') distributions, i.e.

$$P(\mathbf{x}|c) = \sum_{i=1}^{K} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c^{(i)}, \boldsymbol{\Sigma}_c^{(i)}) w_c^{(i)} \qquad \text{where} \qquad \forall c : \sum_i w_c^{(i)} = 1. \qquad (5)$$

Here $\boldsymbol{\mu}_c^{(i)}$ and $\boldsymbol{\Sigma}_c^{(i)}$ are the mean and covariance of the i-th mixture component for class c, and $w_c^{(i)}$ the component's weight. Fitting such a distribution on training samples is typically done using the Expectation-Maximization (EM) algorithm which iterates between optimizing the weights, and optimizing the K Normal distributions.

In this course, we will not investigate the EM algorithm, and you are not required to know how it works. For training a Gaussian-Mixture-Model, the built-in `gmdistribution` class in the Matlab Statistical Toolbox can be used, which uses EM internally.

D Receiver Operating Characteristic (ROC) curves

Instead of reporting a single test error value for a fixed decision threshold, one can also consider changing the threshold to trade-off different types of classification error. Lowering the threshold results in more test cases being assigned to the positive (e.g. pedestrian) class, while increasing it results in more samples classified negatively (e.g. non-pedestrian). This trade-off is then reflected in the False Positive Rate (FPR) and True Positive Rate (TPR), both of which are numbers between 0 and 1.

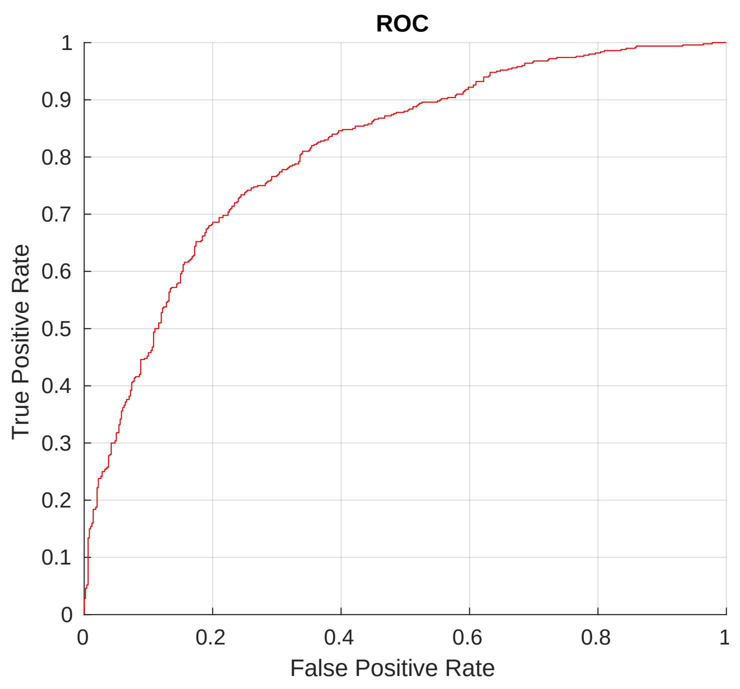

Figure 3: Example of an ROC curve for a certain classifier. We can see for instance that we obtain a True Positive Rate of about 70% (e.g. actual pedestrians classified as pedestrians) at a False Positive Rate of 20% (e.g. actual non-pedestrians classified as pedestrians).

Let d_i be the classifier's decision value for test sample i, and $y_i \in \{-1, +1\}$ the true class label of the sample. Also, let the function $\text{count}_i[x(i)]$ count the number of samples for which condition x holds. Then the TPR and FPR for a given threshold τ are expressed as,

$$P = \text{count}_i[y_i \geq 0] \qquad \text{number of samples in positive class} \tag{6}$$

$$N = \text{count}_i[y_i < 0] \qquad \text{number of samples in negative class} \tag{7}$$

$$TP(\tau) = \text{count}_i[(d_i \geq \tau) \wedge (y_i \geq 0)] \qquad \text{number of positive samples classified as positive} \tag{8}$$

$$FP(\tau) = \text{count}_i[(d_i \geq \tau) \wedge (y_i < 0)] \qquad \text{number of negative samples classified as positive} \tag{9}$$

$$TPR(\tau) = \frac{TP(\tau)}{P} \qquad FPR(\tau) = \frac{FP(\tau)}{N} \qquad \text{true and false positive rates.} \tag{10}$$

By computing the TPR and FPR for varying thresholds, one can create a so-called Receiver-Operating Characteristic (ROC) curve. Figure 3 shows an example of an ROC plot for a single classifier. The ROC curve always starts at (0,0), which corresponds to setting the threshold so high that all test samples are assigned to the negative class, hence there would be no false positives, but also no true positives. The other extreme is obtained when the threshold is so low that everything is assigned to the positive class, in which case both FPR and TPR become one as the curves reaches the top-right corner at (1,1). The curve of an ideal classifier would touch the top-left corner, corresponding to TPR of 1 for a FPR of 0.