Theory question examples

Theory - question example 1
Describe and compare Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA), covering the following aspects:

- Goals of the two models and their formulation
- Training objective of the two models
- Characteristics of the PCA principal components and of the LDA discriminant directions
- How the models can be employed in classification tasks

Theory - question example 2

Considering the Linear Discriminant Analysis (LDA) approach for binary classification and the Tied MVG binary classifier, detail:

- Model formulation, training objective and inference procedure (i.e. how to employ the model for classification) of the LDA classifier
- Model assumptions, training objective and inference procedure of the Tied MVG classifier
- The relationship between the two models
- The form of the decision rules of LDA and Tied MVG binary classifiers

For multiclass problems, LDA can be employed as a dimensionality reduction technique. In this context, briefly explain the objective function of LDA and the limitations of the approach.

Theory - question example 3
Describe in detail the multivariate Gaussian classifier, covering the following aspects:

- Model assumptions
- Estimation of the model parameters
- How the model can be employed to perform inference (i.e. classify a test sample) for both multi-class and binary problems
- The form of decision rules for binary problems
- Naive Bayes and Tied Covariance variants of the model, focusing on
  - Differences with the standard (unconstrained) model in terms of assumptions and decision rules
  - Benefits and limitations with respect to the unconstrained model

Theory - question example 4
Describe the binary logistic regression model for classification, covering the following aspects:

- Classification rule of the binary logistic regression model
- Probabilistic interpretation of the model and of the classification score
- Estimation of the model parameters and possible interpretations of the training objective function

Both logistic regression and Support Vector Machines (SVM) can be interpreted as risk minimization approaches.

- Compare the objective functions of the two models
- Explain possible approaches to obtain non-linear decision functions with these two classifiers

Theory - question example 5

Describe the Support Vector Machine classifier, covering the following aspects:

- Classification rule of SVM and interpretation of the SVM score
- The concept of margin
- Primal (both constrained convex quadratic programming and hinge loss) and dual formulation of the objective function, and relationship between the primal and dual solutions
- SVMs for non linear classification

Both logistic regression and Support Vector Machines (SVM) can be interpreted as risk minimization approaches.

- Compare the objective functions of the two models

Theory - question example 6
Describe Gaussian mixture models in the context of density estimation and pattern classification, covering the following aspects:

- Definition of the model, interpretation of the model parameters and formulation of the GMM as a latent variable model
- Estimation of the model parameters
- How the model can be used to solve classification problems, including open-set classification tasks
- Potential issues of GMMs, possible ways to address these issues, and possible variations of the model

Project questions

Project - question example 1
Explain, in light of the characteristics of the classifiers and of the characteristics of the project datasets:

1. The relative performance of the MVG, Tied MVG and GMM models.
2. The relative performance of linear and non-linear SVM.

Project - question example 2
Explain the relative performance on the project validation set of different SVM kernels (including linear models), in light of the characteristics of the kernel and the characteristics of the dataset. Briefly analyze the effects of regularization on the model performance.

Project - question example 3
Consider the SVM and logistic regression classifiers. In lights of the characteristics of the datasets and of the classifiers, explain the gap between minimum and actual DCF for each model, and, if necessary, the method that you employed to reduce this gap for the project dataset. Analyze also the effects of regularization on the miscalibration error for both models.

Project - question example 4
Given the following functions (assume these functions are already implemented unless specified):

- `trainPCA`: trains a PCA model
- `applyPCA`: applies a PCA model to some data
- `trainClassifier(D, L)`: trains a given classifier from the data matrix `D` and the label vector `L`; returns an object containing the trained model parameters
- `scoreClassifier(clsModel, D)`: computes the array of scores for classifier `clsModel` (as returned by the function `trainClassifier`) for the samples in data matrix `D`
- `evaluateScores(S, L)`: computes a performance metric (e.g. minimum DCF) over the score array `S` with label vector `L`

a) Provide a possible signature and an implementation of the functions `trainPCA` and `applyPCA`, briefly explaining also the function parameters and the return value.

b) Using these functions, write the Python code to:

- Train the classifier on a training set, optimizing the PCA dimension with respect to the provided metric function using a single-fold cross-validation approach
- Evaluate its performance on an evaluation set.

Assume that you have at your disposal a training set, already divided in model training data `(DTR, LTR)` and validation data `(DVAL, LVAL)`, and an evaluation set `(DTE, LTE)`. `DTR`, `DVAL` and `DTE` are data matrices, with samples organized as column vectors, whereas `LTR`, `LVAL` and `LTE` are arrays containing the corresponding labels. To select the PCA dimension m consider all possible values of m that are compatible with the dimension of the feature vectors. Assume that the classifier is invariant to affine transformations, that it does not include hyper-parameters to tune, and that PCA is the only kind of pre-processing to analyze.

Summary of main `numpy` (`np`) and `scipy` employed in the laboratories

`s, U = np.linalg.eigh(C)`
returns the array of eigenvalues `s` in ascending order and the matrix of corresponding eigenvectors `U` of a real symmetric matrix `C`

`U, s, Vh = np.linalg.svd(C)`
returns the array of singular values `s` in descending order, the corresponding matrix of left singular vectors `U` and the corresponding transposed matrix of right singular vectors `Vh`

`s, v = np.linalg.slogdet(C)`
returns the sign `s` and the logarithm of the absolute value `v` of the determinant of matrix `C`

`v = scipy.special.logsumexp(M, axis=k)`
computes in a numerically more stable way `np.log(np.sum(np.exp(a), axis=k))`

`v = np.logaddexp(a, b)`
computes in a numerically more stable way `np.log(np.exp(a) + np.exp(b))`

Project - question example 5
You are given the following functions (assume these functions are already implemented unless specified):

- `trainRBFKernelSVM(D, L, C, gamma)`: trains an SVM model with an RBF kernel with hyper-parameter `gamma` and returns an object containing the trained model information; `D` is the training data matrix, `L` is the corresponding label array, and `C` is the SVM cost-vs-margin trade-off coefficient
- `scoreRBFKernelSVM(svmModel, D)`: computes the classification scores for samples in the data matrix `D` for an SVM model `svmModel` (as returned by the function `trainRBFKernelSVM`) and returns an array of scores
- `evaluateScores(S, L)`: computes an evaluation metric (e.g. minimum DCF) over the array of scores `S` with associated array of labels `L`

Assume that you have at your disposal a training set, already divided in model training data `(DTR, LTR)` and validation data `(DVAL, LVAL)`, and an evaluation set `(DTE, LTE)`. `DTR`, `DVAL` and `DTE` are data matrices, with samples organized as column vectors, whereas `LTR`, `LVAL` and `LTE` are arrays containing the corresponding labels.

Write the Python code to train and apply an SVM classifier. In particular, the code should

- Train an SVM classifier, optimizing the value of the hyper-parameters with respect to the metric function `evaluateScores` using a single-fold cross-validation approach.
- Evaluate the selected SVM model on the evaluation data, using the provided metric.

Write an implementation of `scoreRBFKernelSVM(svmModel, D)`. Assume that `svmModel` is an object with the following fields:

`sv`: `numpy` 2-D array of support vectors, stored as column vectors

`alpha`: Lagrange multiplier values associated to each sv, as a 1-D `numpy` array

`labels`: 1-D `numpy` array of labels (+1 or -1) associated to the support vector

`gamma`: RBF kernel hyper-parameter γ

You can assume that you have at your disposal a function `RBFKernel(D1, D2, gamma)` that returns the matrix of kernel values $k(\mathbf{x}, \mathbf{y})$ for all pairs of samples $\mathbf{x}, \mathbf{y}$ of 2-D sample matrices `D1`, `D2` (i.e., if `K = RBFKernel(D1, D2, gamma)`, then `K[i, j]` is the kernel between arrays `D1[:, i]` and `D2[:, j]`).

Project - question example 6

Consider a binary classification problem, with classes labeled as 1 and 0, respectively.

Let `(DTR, LTR)`, `(DVAL, LVAL)` represent a labeled training set and a labeled validation set. `DTR` and `DVAL` are 2-D numpy arrays containing the dataset samples (stored as column vectors), whereas `LTR` and `LVAL` are 1-D numpy arrays containing the sample labels. Let also `DTE` represent the dataset matrix (again, a 2-D numpy array) containing the samples that our application should classify.

Write a Python code fragment that:

1. trains a *calibrated* binary classifier
2. performs inference (i.e. computes predicted *labels*) on the evaluation data

You can assume that the following functions have been defined:

- `trainClassifier(D, L)`: train a non-calibrated classification model (e.g., an SVM or an LDA classifier) on the training matrix `D` with associated labels array `L`, and return a python object containing the trained model (assume that the model does not contain tunable hyper-parameters)
- `scoreClassifier(model, D)`: compute the non-calibrated classification scores for model `model` (as returned by `trainClassifier`) for the samples in data matrix `D` and return a 1-D array of scores
- `trainCalibrationModel(S, L, prior)`: train a calibration model on the 1-D array of scores `S`, with associated array of labels `L`, for a binary application with prior `prior` for class 1, and return a python object containing the trained model
- `applyCalibrationModel(calModel, S)`: apply the calibration model `calModel` (as returned by `trainCalibrationModel`) to the 1-D array of scores `S`, and return a 1-D array of calibrated scores

NOTE: assume that the target application is characterized by an effective prior `p` for class 1. You are not required to tune the calibration model hyper-parameter `prior`, but you can assume that the calibration model can be trained using the target application prior `p`.