Multi-Class Classification with Kernel-based Latent Factor Models and Perceptrons

CS342: Machine Learning, Dept. of Computer Science, University of Warwick, UK

I. Introduction

The coursework focuses on designing and training a multi-class perceptron to classify a real-world dataset. The perceptron classifier should use high-dimensional data, which should be generated by applying a kernel-based latent factor model to the original data set.

Manifold learning is widely used in machine learning to project data onto a linear or non-linear representation of usually smaller dimensions. Such a technique is commonly used as a pre-processing step for clustering or classification algorithms to reduce the dimensionality and computational costs, and improve performance. This technique is also widely used for feature extraction. Latent factor models are then part of the realm of manifold learning.

Recall that for dimensionality reduction, given a training dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ with n data points of d dimensions, we would like to find a lower-dimensional embedding $\mathbf{Z} \in \mathbb{R}^{n \times k}$, where k denotes the dimensions of the new embeddings and $k \ll d$. In class, we covered Principal Component Analysis (PCA) as a powerful **linear** manifold learning.

In some cases, projecting the data onto a higher dimensional subspace may improve the performance of classification tasks, as manifold learning can discover a more powerful representation for the data. In such cases, we have $k \gg d$. This coursework involves such a projection by using a kernel-based latent factor model that introduces non-linearities in the projection.

A. Dataset

The dataset for this coursework is the Iris Dataset from the UCI Machine Learning Repository, which is available on the module webpage. To learn more about this dataset, see https://archive.ics.uci.edu/ml/datasets/iris. The dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant. The classes are: {"Iris Setosa," "Iris Versicolour," "Iris Virginica"}. One class is linearly separable from the other two, while the latter are not linearly separable from each other. Each instance is represented by a 4d feature vector with 4 features:

1) sepal length in cm
2) sepal width in cm
3) petal length in cm
4) petal width in cm

II. Methods

A. Data visualization

Use PCA to visualize the data points of the Iris Dataset in 2 dimensions (2d). This requires to project the data onto the two top principal components (PCs). Determine which classes are linearly separable and which are not based on the 2d embeddings. You may use your implementation of PCA based on SVD from Lab 5.

B. Multi-class classification using a perceptron

Design a multi-class perceptron to classify the data in the 2d subspace defined by PCA. Note that Lab 6 involves implementing a two-class perceptron. You may use that implementation as a starting point to implement your multi-class perceptron. To implement your multi-class perceptron, please recall the ideas discussed in Lecture 14 for multi-class linear classification and how it is important to encourage the largest predicted numerical value given by the set of linear classifiers to be the correct prediction.

Additional hints:

1) Define the class labels as numerical values, e.g., use 0, 1 and 2, for the three labels.
2) Determine how to update the weights of the linear classifier that provides the maximum predicted value and those of the linear classifier associated with the correct class when the prediction is incorrect.
3) Recall adding a bias to the linear classifiers.
4) The algorithm may need to run for several iterations, as the data is not linearly separable. In each iteration, all training samples should be used in random order. Train your multi-class perceptron for a total of 50 iterations and use the accuracy attained at iteration 50 as the final accuracy.

Using your implementation of the multi-class perceptron, compare and discuss if projecting of the data onto a 2d subspace as defined by PCA improves classification performance compared to the case of using the multi-class perceptron on the original data. Note that this requires training two multi-class perceptrons, one for the original data and one for the PCA-projected data.

Use your multi-class perceptron to attempt to fully separate the data when all four PCs are used to project the data. Note that this requires training an additional multi-class perceptron. Discuss if the classification improves when all PCs are used for projection. Recall that all training processes should be run for 50 iterations and the accuracy attained at iteration 50 is the final accuracy.

C. Projection onto a high-dimensional subspace

To more accurately classify the data points of the Iris Dataset, we will use a Kernel to define a non-linear manifold learning technique. Specifically, we will create a non-linear version of PCA to project the data onto a high-dimensional subspace. To this end, a normalized Kernel matrix, $\bar{\mathbf{K}}$, must be first computed. Lecture 16 explains how to create a Kernel matrix, $\mathbf{K}$, for a dataset, $\mathbf{X}$. To normalize $\mathbf{K}$, use:

$$\bar{\mathbf{K}} = \mathbf{K} - \mathbf{AK} - \mathbf{KA} + (\mathbf{AK})\mathbf{A}, \tag{1}$$

where $\mathbf{A}$ is a square matrix with the same dimensions as $\mathbf{K}$ and all values equal to $\frac{1}{n}$, where n is the number of training samples in $\mathbf{X}$.

This coursework focuses on the radial basis function kernel, or RBF kernel, given by the following expression:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2\right), \tag{2}$$

where $\gamma = \frac{1}{2\sigma^2}$ and σ is the width or scale of a Gaussian function centered at $\mathbf{x}_j$.

A non-linear version of PCA can be defined by finding the PCA decomposition of the normalized matrix $\bar{\mathbf{K}}$. Notice that γ is a hyper-parameter that must be manually defined for the RBF kernel. For high-dimensional data, it is common practice to do a grid search on a range of parameters to determine the best value.

What is a grid search? A grid search is a process that searches exhaustively through a manually specified subset of the hyper-parameter space of the targeted algorithm. Although a validation set is commonly used to do the grid search, for the case of manifold learning, it is common practice to use the entire training dataset, as we are interested in finding important characteristics of the underlying structure of the data. From a grid search, one can draw many conclusions. For example, one can determine the appropriate number of PCs to project the data, and the impact that very large and very low hyper-parameter values have on the task at hand. In our case, the task at hand is multi-class classification.

Based on a grid search, determine the appropriate number of PCs and value for hyper-parameter γ based on the classification accuracy attained by the multi-class perceptron when using the data projected by non-linear PCA. Since we are interested in a high-dimensional subspace, test a range of PCs from a value greater than d (recall that d denotes the dimensions of the data) to the maximum number of PCs of the nomalized kernel matrix, $\bar{\mathbf{K}}$. Use increments of 1. For γ, the range to test can be roughly determined by the variance of the features after centering the data, $\mathbf{X}$. To this end, compute the variance of each feature, i.e., each column of $\mathbf{X}$ after centering it. Use a range of values that fully cover the range of values for the variances found. For example, if the minimum variance found is $\sigma^2_{min} = 0.5$ and the maximum is $\sigma^2_{max} = 1.5$, the range of values to test for γ should be (recall that $\gamma = \frac{1}{2\sigma^2}$):

$$\left[\gamma_{min} = \frac{1}{2(\sigma^2_{max} + \epsilon)}, \gamma_{max} = \frac{1}{2(\sigma^2_{min} - \epsilon)}\right], \tag{3}$$

where ϵ is a small constant that guarantees that the full range of variances of the data is accounted for. In our example, with $\epsilon = 0.05$, the range is $[0.32, 1.11]$. Use $\epsilon = 0.05$ to determine the range of values of γ for the Iris Dataset. To test that range of values, use increments of 0.10.

Plot the results of the grid search as a matrix (in your Jupyter notebook) with different colors denoting the classification accuracy of the trained multi-class perceptrons. A sample matrix is shown in Fig. 1, where the color of each cell indicates the number of errors made by the corresponding multi-class perceptron (red: maximum no. of errors, light yellow: minimum number of errors). Recall using a maximum of 50 iterations to train a multi-class perceptron. For each trained multi-class perceptron, use the accuracy attained at iteration 50. To select the best hyper-parameter value and number of top PCs, our objective is to classify the data as accurately as possible while keeping the size of the model, in terms of the number of parameters learned, as small as possible.

Discuss and justify the selected hyper-parameter value and number of top PCs to use with the RBF kernel. Determine if the data can be linearly separated in a high-dimensional subspace.

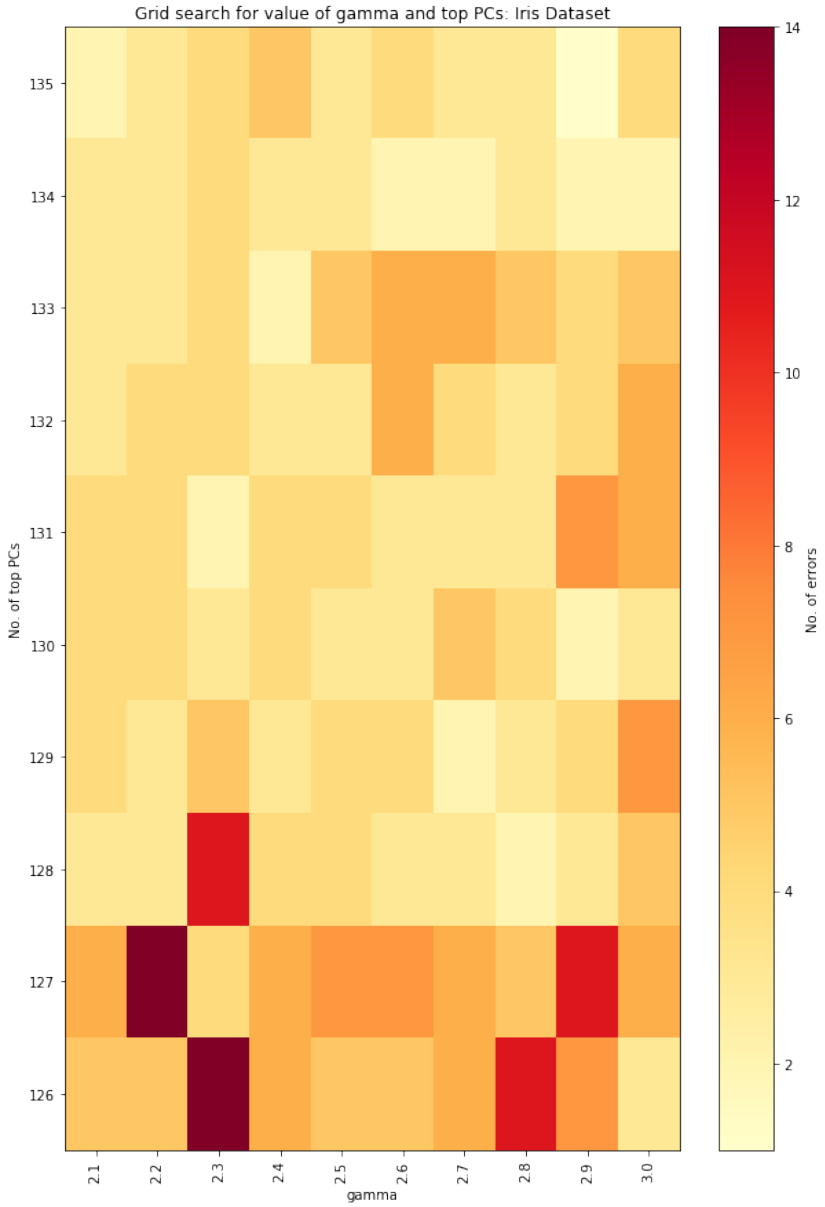

Fig. 1: Sample matrix showing the no. of errors made by several multi-class perceptrons trained using projected data (non-linear PCA) with a specific number of top PCs and a value of γ for the RBF kernel. The color of each cell indicates the number of errors made by the corresponding multi-class perceptron (red: maximum no. of errors, light yellow: minimum number of errors)

III. Deliverables

Submit the following via Tabula:

1) A report as a PDF with the figures and discussions requested (you may use snippets of your code in your report to explain results).
2) Your code as a Jupyter file. Please make sure that the submitted code works correctly and is well-organized and commented.

The breakdown of marks is as follows:

- Section II.A: **5 marks**.
- Section II.B:
  - Implementation of multi-class perceptron: **35 marks**.
  - Results and discussions: **10 marks**.
- Section II.C:
  - Implementation of normalized kernel matrix: **20 marks**.
  - Matrix showing the results of the grid search: **15 marks**.

- Discussions and justification of selected hyper-parameter value and number of top PCs: **5 marks**.

A total of **10 marks** are available for the quality and presentation of the report, as well as the organization and explanation of your code (in the form of comments added within the code). Documents should be clearly and logically structured, well-written and adequately proof-read before submission. To structure your document, follow the structure of *Section II. Methods*. The suggested length is between 1200-1500 words. The standard department late penalties and plagiarism policies are in effect. Note that there are several ready-to-use implementations of kernel-based latent factor models, including PCA, the perceptron algorithm, and grid searches in Python. You must implement your own solutions and **must not** use these libraries. Your code from Labs 5 and 6 may be used as a starting point to complete this coursework.