CS342 Multi-Class Classification with Kernel-based Latent Factor Models and Perceptrons
Report

u1835495

Department of Computer Science
University of Warwick
2020-21

Contents

1 I.A Dataset

To begin with, the Iris Dataset is provided for the coursework containing 3 types of Iris plants with 50 elements for each. The classes of plants are *Iris Setosa*, *Iris Versicolour* and *Iris Virginica*. Each instance in the dataset contains 4 features representing length and width in cm of sepal and petal of the plant. The data is given within *iris.data* file.

To handle the data provided and perform calculations Python *NumPy* library is used. Data file has to be uploaded to *NumPy* arrays. The *iris.data* file is stored in the /data directory in the same directory as *coursework.ipbn*. Location of the file is specified in the *location* string variable which can be changed regarding the location of the data file. The features of all the instances, first fours columns of dataset, are uploaded to *iris_x* NumPy array. Then the column of names of each element is loaded to *Pandas* dataframe and after defining class labels as numerical values with 0, 1 and 2, labels are stored in NumPy array.

2 II.A Data visualization

This part of the coursework asks to visualize the data in 2 dimensions. The data given is in 4 dimensions (each element contains 4 features), hence, the Principle Components Analysis is used to visualize data in 2 dimensions. Using Principle Components Analysis (PCA) for the data Principle Components (PCs) are computed that can be used to change basis (in this case, reduce) on the data when using top PCs and discarding the remaining components.

To compute the Principal Components the data is first centered using NumPy *mean()* on the *iris_x* data. Then on the centered data Singular Value Decomposition is performed using NumPy *numpy.linalg.svd* that decomposes the matrix of our data and produces another matrix V where each row of this matrix contains a Principal Component. From the matrix produced 2 first rows are taken (represents top 2 Principal Components) and the initial data is then projected onto the 2 top PCs. The data projected into 2 dimensions is then used to make a visual representation of the Iris Dataset and its classes using *matplotlib* Python library (Figure 1).

Now with the data points of the Iris Dataset in 2 dimensions visually represented, it can be clearly seen that Iris-setosa class can be linearly separated from other two classes - Iris-versicolor and Iris-virginica. As well as, it also confirms that the Iris-versicolor and Iris-virginica are not linearly separable from each other.

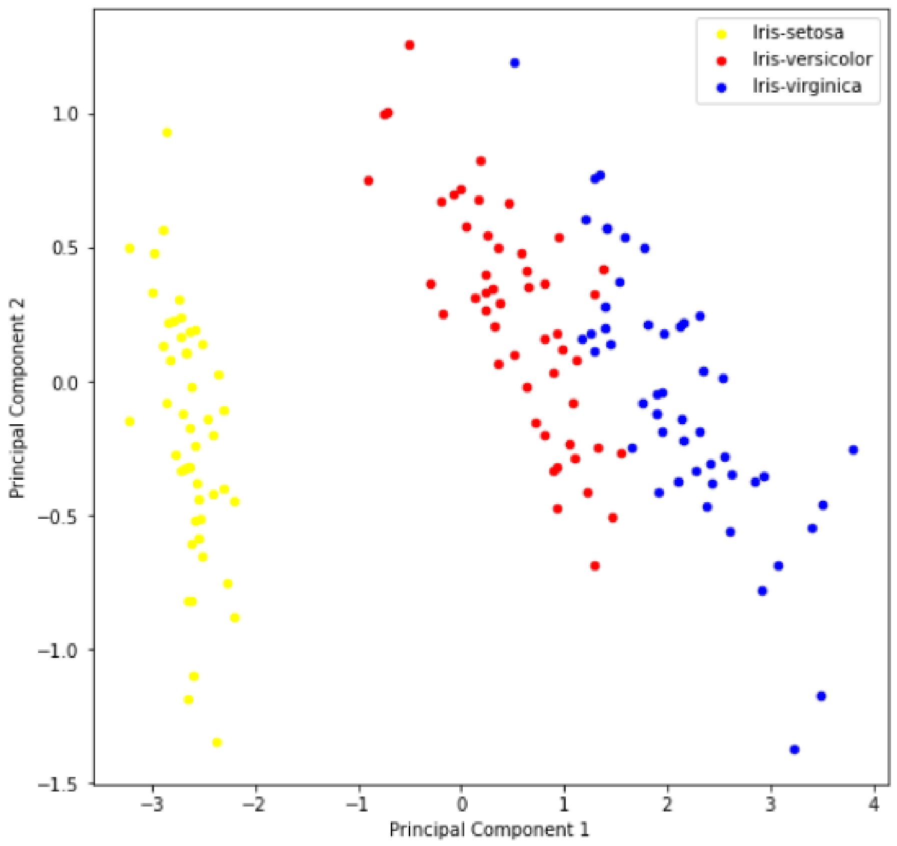
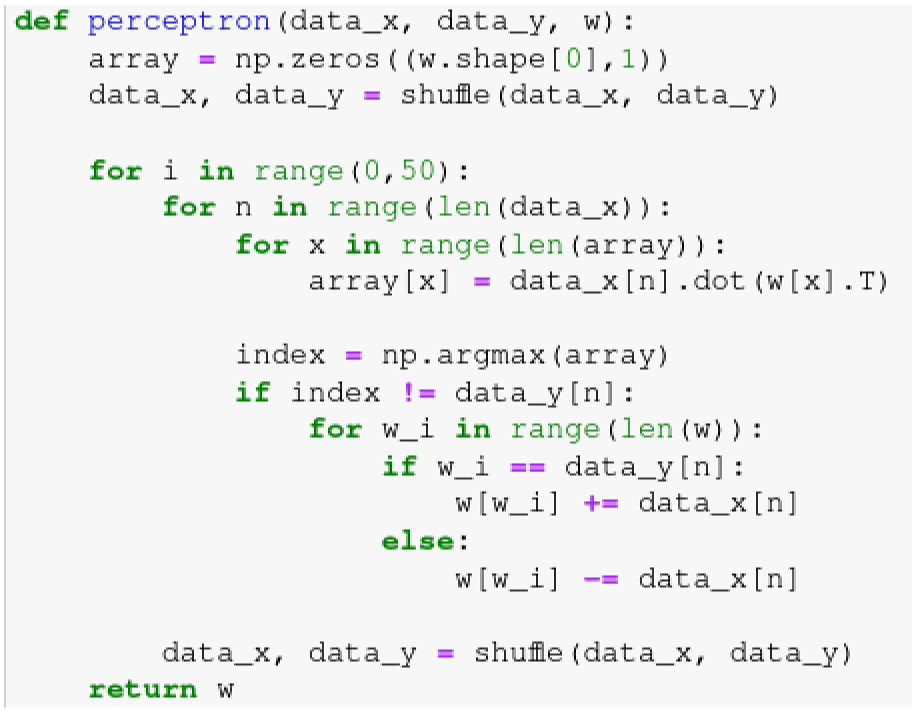


Figure 1: Iris Dataset in 2 dimensions

3 II.B Multi-class classification using a perceptron

For the part II.B of the coursework multi-class perceptron is implemented. The perceptron is an algorithm of binary classifiers for linearly separable data. However, the Iris Dataset is not linearly separable. Yet with some slight changes the algorithm can be transformed into a multi-class perceptron for linearly non-separable data. Three multi-class perceptrons are trained in this part. First, a perceptron to classify the data projected onto 2 dimensions, also a perceptron is trained on the original data and third perceptron to classify the data projected onto 4 dimensions using all four Principal Components.

Method *perceptron()* is implemented. As the arguments to this method are given training data as *data_x*, labels for each instance in training set as *data_y* and initial weight vector of zeros for each class - w. Since the Iris Dataset is not linearly separable, algorithm runs for 50 iterations and, for the same reason, single iteration contains testing all training examples in the dataset. Just like the perceptron of binary classifier use *sign()* function to classify instances into +1 and -1 classes this algorithm is using NumPy *argmax()* method to classify an example into one of the three classes - 0, 1 and 2. The interim array is used to store three values of an example multiplied by each class weight vector. The *argmax()* classifies training example by returning the index of the greatest value in array, if the example is misclassified, weight vectors has to be refined. When updating the vectors it is important to encourage the largest predicted value for the correct class and the smallest value for the incorrect classes. Thus, weight vector of the correct class is increased by the example value while weight vectors of incorrect classes are decreased. The dataset is then shuffled again for the next iteration. After 50 iterations weight vectors of trained multi-class perceptron are returned (Implementation of the algorithm in Figure 2).

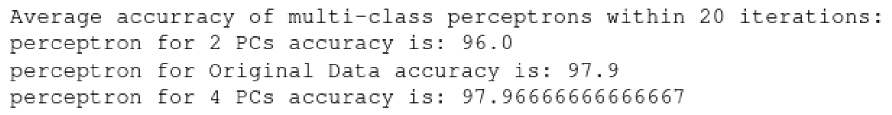


```
def perceptron(data_x, data_y, w):
    array = np.zeros((w.shape[0],1))
    data_x, data_y = shuffle(data_x, data_y)

    for i in range(0,50):
        for n in range(len(data_x)):
            for x in range(len(array)):
                array[x] = data_x[n].dot(w[x].T)

            index = np.argmax(array)
            if index != data_y[n]:
                for w_i in range(len(w)):
                    if w_i == data_y[n]:
                        w[w_i] += data_x[n]
                    else:
                        w[w_i] -= data_x[n]

        data_x, data_y = shuffle(data_x, data_y)
    return w
```

Figure 2: Perceptron implementation

Another method that is implemented in this part is *accuracy()*. The method given as the arguments *w* of trained multi-class perceptron, data to be classified *data_x* and correct labels for each instance *data_y* returns the percentage of the accuracy of trained model.

Having these two methods implemented, performance of different multi-class perceptrons can be compared. The train datasets are prepared for each case by adding a bias column to the data projected onto 2 dimensions, 4 dimensions and original data. For each case initial weight vectors are created of 3 rows and n columns where n is a number of features in test data. Now for each case a perceptron is trained and the accuracy of each multi-class perceptron is calculated. This is done within 20 iterations calculating the average accuracy of each perceptron (possible results of such calculation in Figure 3).



```
Average accurracy of multi-class perceptrons within 20 iterations:
perceptron for 2 PCs accuracy is: 96.0
perceptron for Original Data accuracy is: 97.9
perceptron for 4 PCs accuracy is: 97.96666666666667
```

Figure 3: Accuracy of different perceptrons

The multi-class perceptron classifying the data in 2d subspace will usually result in approximately 96% accuracy. However, using multi-class perceptron on original data or on data projected on all 4 Principal Components results in better accuracy, each classifies approximately 98% of examples correctly. Most of the time the latter will perform slightly better, however, the perceptron on original data sometimes can also perform better between the two.

4 II.C Projection onto a high-dimensional subspace

This part of the coursework focuses on classifying the Iris Dataset points even more accurately. The non-linear manifold learning technique defined by Kernel is used to project the data onto high-dimensional space by creating a non-linear version of Principal Component Analysis. To do that normalized Kernel matrix $\bar{K}$ is calculated using Kernel matrix **K** and a square matrix **A**. To calculate **K**, kernel function $k(x_i,x_j)$ is given that is the radial basis function. Also a grid search is performed to determine the appropriate hyper parameters for the algorithm, to be specific, number of top PCs and γ that is used for the kernel matrix **K** calculation.

First, the $\gamma_{\min}$ and $\gamma_{\max}$ is calculated using the variance of the features in the centered Iris Dataset. Then the for loop starts iterating over a range of PCs starting from 5 which is greater than number of dimensions of the data. Inner loop then iterates over gammas from $\gamma_{\min}$ to $\gamma_{\max}$ with increments of 0.1. Now for each number of PCs and each gamma, $\bar{K}$ is calculated by calling method *k_tild()* that given the centered data and γ calculates the $\bar{K}$ normalized kernel matrix and returns it. The Singular Value Decomposition is performed on the normalized matrix and number of PCs as the value in the outer loop is selected to project the data into k dimensions. The bias column is being added to the projected data, initial class vectors are created and the multi-class perceptron on k-dimensional data is being trained. Once it is trained, the accuracy of the model is calculated. If the perceptron is perfect (accuracy - 100%) the search is stopped and the hyper parameters of the first perfect perceptron are selected.

A piece of the search grid is displayed which contains the perfect perceptron and displays the number of errors made by perceptrons with adjacent hyper parameters (Figure 4).

The selection of hyper parameters focuses on keeping the size of the model as small as possible while performing as accurate as possible. For that reason, the grid search is iterating through the PCs number in increasing order and stops when the first perfect multi-class perceptron is found which applies for both constraints - to keep a small size of the model and have accurate predictions. Since now the predictions can get as accurate as 100%, the data can be, for some multi-class perceptron be linearly separable in high-dimensional subspace.

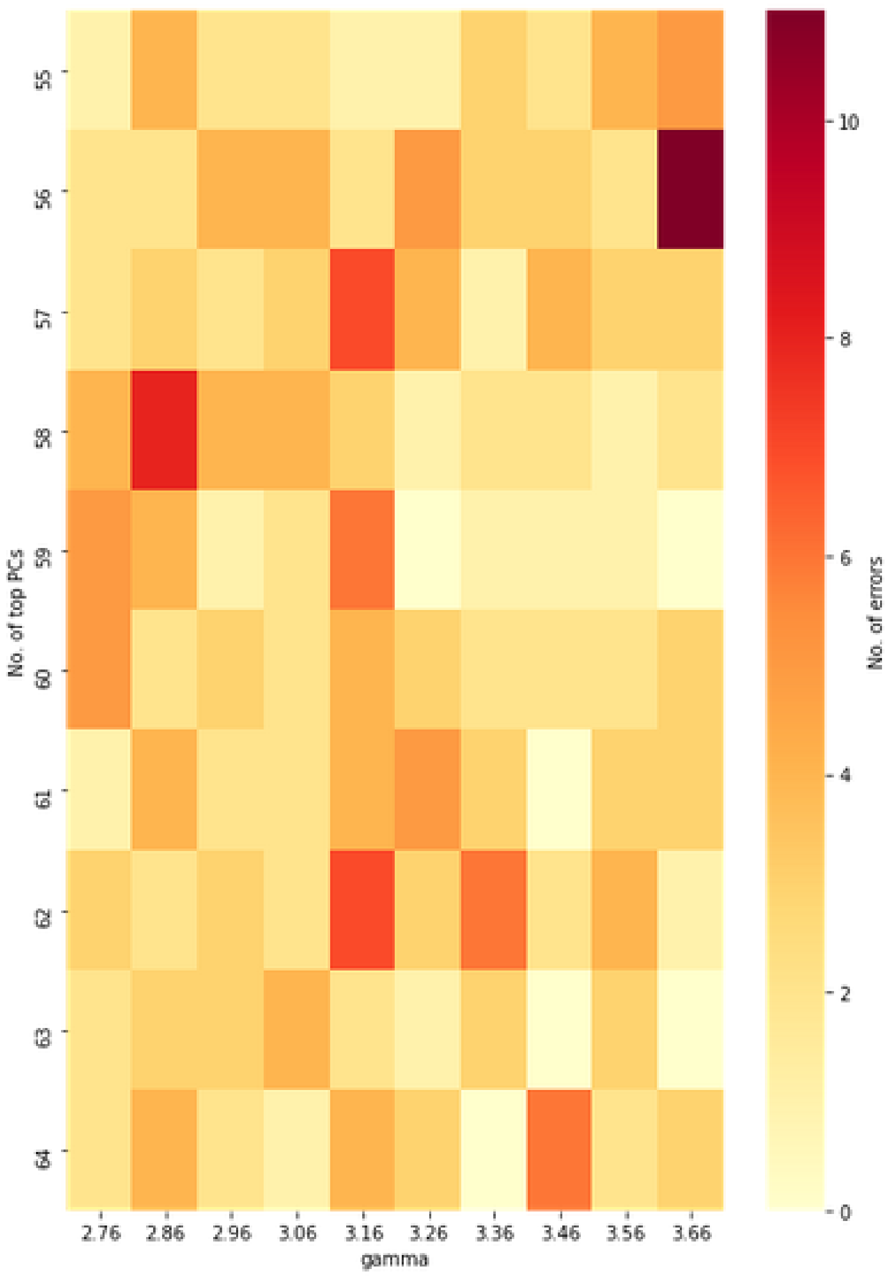


Figure 4: Matrix showing number of errors made by several multi-class perceptrons. In this case, the selected parameters for the perfect perceptron are - 59 PCs and gamma with value 3.26.