Machine Learning

Summer Semester 2019, Homework 3

Prof. Dr. J. Peters, H. Abdulsamad, S. Stark, D.Koert

TECHNISCHE UNIVERSITÄT DARMSTADT

Total points: 67 + 10 bonus
Due date: Friday, 5 Juli 2019 (17:00)
You need to hand in the pdf in moodle **and** a printed version to the postbox from the IAS secretary office (S2 02 | E315)

Group ID : 146
Name, Surname, ID Number **Peter Nickl, 1941346** **Steffen Schï¿œfer, 2635897**

Problem 3.1 Linear Regression [28 Points + 5 Bonus]

In this exercise, you will use the dataset `linRegData.txt`, containing 150 points in the format <`input variable, output variable`>. The input is generated by a sinusoid function, while the output is the joint trajectory of a compliant robotic arm. The first 20 data points are the training set and the remainder are the testing set.

a) **Polynomial Features [10 Points]**

Write the equation of the model and fit it with polynomial features. Using the Root Mean Square Error (RMSE) as a metric for the evaluation, select the complexity of the model (up to a 21st degree polynomial) by evaluating its performance on the testing data. Which is the best RMSE you achieve and what is the model complexity? Does it change if we evaluate our model on the training data? Comment your findings and plot the RMSE for each case (use two lines, one for evaluation on training data, one for evaluation on testing data). For the estimation of the optimal parameters use a ridge coefficient of $\lambda = 10^{-6}$.
Using what you think is the best learned model from the previous point, show in a single plot the ground truth (full dataset) and the model prediction over it. Attach snippets of your code showing how you generate polynomial features and how you fit the model.

b) **Gaussian Features [4 Points]**

Now use Gaussian features. Each feature is a Gaussian distribution were the means are distributed linearly in $x \in [0, 2]$ and the variance is set to $\sigma^2 = 0.02$. The features have to be normalized, i.e., they have to sum to one at every x. Using 10 features generate a plot with the activation of each feature over time (i.e., plot the matrix Φ). Attach a snippet of your code showing how to compute Gaussian features.

c) **Gaussian Features, Continued [4 Points]**

Repeat the process of fitting the model using the Gaussian features from the previous question. Compare the RMSE on the testing data using $15 \dots 40$ basis functions and plot the RMSE. Which number of basis functions has the best performance and what is the best RMSE? Use a ridge coefficient of $\lambda = 10^{-6}$.

d) **Bayesian Linear Regression [10 Points]**

Using Bayesian linear regression, plot the mean and the standard deviation of the predictive distribution learned using the first $\{10, 12, 16, 20, 50, 150\}$ data points (one plot per case; plot it in the interval $x \in [0, 2]$). Discuss how the model uncertainty changes with the amount of data points and the problem of overfitting with Bayesian linear regression. Use the best performing polynomial features that you found in 3.1a, a ridge coefficient of $\lambda = 10^{-6}$, and assume Gaussian noise with $\sigma^2 = 0.0025$.

e) **Cross Validation [5 Bonus Points]**

So far, we have split our dataset in two sets: training data and testing data. Cross-validation is a more sophisticated approach for model selection. Discuss it and its variants, pointing out their pro and cons.

Problem 3.2 Linear Classification [16 Points]

In this exercise, you will use the dataset `ldaData.txt`, containing 137 feature points $\boldsymbol{x}$. The first 50 points belong to class C_1, the second 43 to class C_2, the last 44 to class C_3.

a) **Discriminative and Generative Models [4 Points]**

Explain the difference between discriminative and generative models and give an example for each case. Which model category is generally easier to learn and why?

Generative models *first use the underlying data to deduce properties of the underlying probability distribution in form of the class-doncitional densties $p(x|C_k)$ for each class C_k. Seperatly they infer the prior class probabilities $p(C_k)$. After that, they use Bayes'theorem in the form*

$$\frac{p(x|C_k)\,p(C_k)}{p(x)} \tag{2}$$

to fint the posteriori class probabilities $p(C_k|x)$. Now decision theory is used to determine class membership.

Since in theory we can use these distribution to create new data, this is called a gernative model. Also note that generative models also work when modeling the joint distribution $p(x, C_k)$ and normalizing.

Discriminative models *determine the posterior class probabilities $p(C_k|x)$ directly. After that decision theory is used to each new input to a class.*

Because generative models need the joint distribution $p(x, C_k)$ for calculating the posterior distribution, instead of deriving it directly, they're much harder to train. This is especially true if x is highly dimensional, since then we need a very big dataset to model class-doncitional densities correctly.

b) **Linear Discriminant Analysis [12 Points]**

Use Linear Discriminant Analysis to classify the points in the dataset, i.e., assume Gaussian distributions in each class with equal covariances and use the posterior distributions for assigning classes. Attach two plots with the data points using a different color for each class: one plot with the original dataset, one with the samples classified according to your LDA classifier. Attach a snippet of your code and discuss the results. How many samples are misclassified? (You are allowed to use built-in functions for computing the mean and the covariance.)

Problem 3.3 Principal Component Analysis [23 Points + 5 Bonus]

In this exercise, you will use the dataset `iris.txt`. It contains data from three kind of Iris flowers ('Setosa', 'Versicolour' and 'Virginica') with 4 attributes: sepal length, sepal width, petal length, and petal width. Each row contains a sample while the last attribute is the label (0 means that the sample comes from a 'Setosa' plant, 1 from a 'Versicolour' and 2 from 'Virginica'). (You are allowed to use built-in functions for computing the mean, the covariance, eigenvalues and eigenvectors.)

a) **Data Normalization [3 Points]**

Normalizing the data is a common practice in machine learning. Normalize the provided dataset such that it has zero mean and unit variance per dimension. Why is normalizing important? Attach a snippet of your code.

In machine learning some methods like PCA take the variance of the features as a criterion for dimensionality reduction. If the feature differ in their range, the means and the variances of the features will have a different scale. To achieve better comparability among the features it is common to normalize the data to unit variance and zero mean. The normalization can be achieved by applying the following formula to all datapoints x_i for each feature separately:

$$\frac{x_i - mean_{feature}}{standard\_deviation_{feature}}$$

We apply the normalization to the dataset according to the following code snippet.

```
import numpy as np
import matplotlib.pyplot as plt
import os

# set the working directory
os.chdir("C:/Users/pistl/Desktop/SML/homework/homework3/")

# print the current working directory
os.getcwd()

# read in data
def load_data():
    data = np.loadtxt(fname =
        "C:/Users/pistl/Desktop/SML/homework/homework3/dataSets/iris.txt",delimiter=',', skiprows=0)
    return data

# calculate mean and standard_deviation
def calculate_stats(data):
    mean = np.mean(data[:,:4],axis=0)
    std_deviation = np.std(data[:,:4],axis=0)
    #eigen = np.linalg.eig(data)

    return mean, std_deviation

# normalize data to zero mean and unit variance
def normalize_data(data, mean, std_deviation):
    data_normalized = ( data[:,:4] - mean ) / std_deviation
    return data_normalized

# read in data
data = load_data()

# calculate mean and standard_deviation
stats = calculate_stats(data)
mean = stats[0]
std_deviation = stats[1]
```

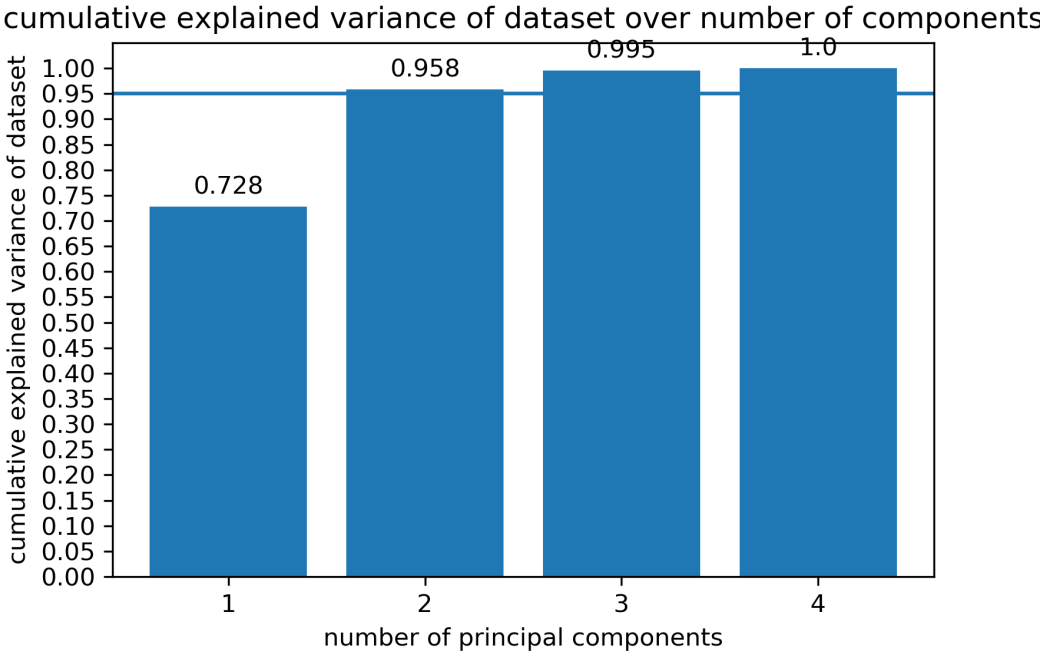

Figure 1: Cumulative explained variance of dataset over number of principal components.

```
# normalize data to zero mean and unit variance
data_normalized = normalize_data(data,mean,std_deviation)
```

b) **Principal Component Analysis [8 Points]**

Apply PCA on your normalized dataset and generate a plot showing the proportion (percentage) of the cumulative variance explained. How many components do you need in order to explain at least 95% of the dataset variance? Attach a snippet of your code.

As Figure 1 shows, by using two principal components we can explain 95,8% of the cumulative variance. The following code snippet shows the code for doing the PCA and for plotting.

```
# do PCA
def PCA(data_normalized):
    cov = np.cov(data_normalized,rowvar=False)
    eigenvalues, eigenvectors = np.linalg.eig(cov)
    explained_var = eigenvalues / sum(eigenvalues)
    return eigenvalues, eigenvectors, explained_var

eigenvalues = PCA(data_normalized)[0]
eigenvectors = PCA(data_normalized)[1]
explained_var = PCA(data_normalized)[2]

# plot explained cumulative variance over number of principal components
fig, ax = plt.subplots()
explained_var_cumulative = np.cumsum(explained_var)
objects = ('1', '2', '3', '4')
y_pos = np.arange(len(objects))
plt.axhline(y=0.95, xmin=0, xmax=4)
plt.xticks(y_pos, objects)
plt.yticks(np.arange(0, 1.05, step=0.05))
plt.ylabel('cumulative explained variance of dataset')
plt.xlabel('number of principal components')
plt.title('cumulative explained variance of dataset over number of components')
rects = ax.bar(y_pos, np.around(explained_var_cumulative, decimals=3))
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3), # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')
autolabel(rects)
plt.savefig('explained_var_cumulative.png',dpi=300)
plt.show()
print(explained_var_cumulative)
```

c) **Low Dimensional Space [6 Points]**

Using as many components as needed to explain 95% of the dataset variance, generate a scatter plot of the lower-dimensional projection of the data. Use different colors or symbols for data points from different classes. What do you observe? Attach a snippet of your code.

d) **Projection to the Original Space [6 Points]**

Reconstruct the original dataset by using different number of principal components. Using the normalized root mean square error (NRMSE) as a metric, fill the table below (error per input versus the amount of principal components used).

N. of components	x_1	x_2	x_3	x_4
1				
2				
3				
4				

Attach a snippet of your code. (Remember that in the first step you normalized the data.)

e) **Kernel PCA [5 Bonus Points]**

Throughout this class we have seen that PCA is an easy and efficient way to reduce the dimensionality of some data. However, it is able to detect only linear dependences among data points. A more sophisticated extension to PCA, *Kernel PCA*, is able to overcome this limitation. This question asks you to deepen this topic by conducting some research by yourself: explain what Kernel PCA is, how it works and what are its main limitations. Be as concise (but clear) as possible.