Machine Learning Worksheet 1

k-Nearest Neighbors and Decision Trees

1 Dataset

You are free to do the following exercises completely by hand or use a computer to help speed things up (python, MATLAB, R, Excel, ...). You should, however, show the basic steps of your work and implement your own "helpers" instead of blindly using code. Using a machine learning toolbox and copying the result will not help you understand.

The table below gives you a feature matrix $\boldsymbol{X}$ together with the targets $\boldsymbol{y}$, where each row i represents one sample. This data is also available on Piazza as `01_homework_dataset.csv`. The column with names for each datapoint i can help you reference specific data points.

i	$x_{i,1}$	$x_{i,2}$	$x_{i,3}$	y_i
A	5.5	0.5	4.5	2
B	7.4	1.1	3.6	0
C	5.9	0.2	3.4	2
D	9.9	0.1	0.8	0
E	6.9	-0.1	0.6	2
F	6.8	-0.3	5.1	2
G	4.1	0.3	5.1	1
H	1.3	-0.2	1.8	1
I	4.5	0.4	2.0	0
J	0.5	0.0	2.3	1
K	5.9	-0.1	4.4	0
L	9.3	-0.2	3.2	0
M	1.0	0.1	2.8	1
N	0.4	0.1	4.3	1
O	2.7	-0.5	4.2	1

2 Decision Trees

Problem 1: Build a decision tree T for your data $\{\boldsymbol{X}, \boldsymbol{y}\}$. Consider all possible splits for all features and use the Gini index to build your tree. Build the tree only to a depth of two! Provide at least the value of the final Gini index at each node and the distribution of classes at each leaf.

Problem 2: Use the final tree T from the previous problem to classify the vectors $\boldsymbol{x}_a = (4.1, -0.1, 2.2)^T$ and $\boldsymbol{x}_b = (6.1, 0.4, 1.3)^T$. Provide both your classification y_a and y_b and their respective probabilities $p(c = y_a \mid \boldsymbol{x}_a, T)$ and $p(c = y_b \mid \boldsymbol{x}_b, T)$

Upload a single PDF file with your solution to Moodle by 30.10.2017, 9:59am CET. The solution must be typeset (using Latex, Word, etc.). Handwritten solutions are not accepted.

3 k-Nearest Neighbors

Problem 3: Load the notebook `01_homework_kNN.ipynb` from piazza. Fill in the missing code and run the notebook. Convert the evaluated notebook to pdf and add it to the printout of your homework.

Note: We suggest that you use Anaconda for installing Python and Jupyter, as well as for managing packages. We recommend that you use Python 3.

For more information on Jupyter notebooks and how to convert them to other formats, consult the Jupyter documentation and nbconvert documentation.

Problem 4: Classify the two vectors $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ given in Problem 2 with the k-nearest neighbors algorithm. Use $k = 3$ and Euclidean distance.

Problem 5: Now, consider y_i to be real-valued targets rather than classes. Perform 3-NN regression to label the vectors from Problem 2.

Problem 6: Look at the data. Which problem do you see w.r.t. building a Euclidean distance-based k-NN model on $\boldsymbol{X}$? How can you compensate for this problem? Does this problem also arise when training a decision tree?

Homework 01 - kNN & Decision Trees

IN2064 Machine Learning

Arne Sachtler - *Registration Number: 03692662*

29th October 2017

1 Problem 1

The tree is built automatically by a written python script. It prints the resulting tree by a textual description:

```
Decision Tree:
    Gini Index: 0.6577777777777777
    Class Distribution: {0: 5, 1: 6, 2: 4}
    Is Leaf: False
    Split at value 4.1 in dimension 0
    Left:
        Gini Index: 0.0
        Class Distribution: {1: 6}
        Is Leaf: True
    Right:
        Gini Index: 0.49382716049382713
        Class Distribution: {0: 5, 2: 4}
        Is Leaf: False
        Split at value 6.9 in dimension 0
        Left:
            Gini Index: 0.4444444444444444
            Class Distribution: {0: 2, 2: 4}
            Is Leaf: True
        Right:
            Gini Index: 0.0
            Class Distribution: {0: 3}
            Is Leaf: True
```

Listing 1: Final decision tree generated from the given exemplary data with a maximum depth of two.

Generating a set of artificial test data and using the generated decision tree in order to predict the class of the generated test data reveals the learned class decision as shown in Figure 1.

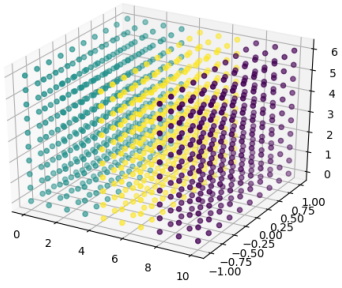


Figure 1: Class prediction for artificially generated test data.

2 Problem 2

The following table summarises the predicted classes for the two given data points.

Data Point x	**Predicted Class** y	**Probability** $p(c = y\|\mathbf{x}, T)$
$\mathbf{x_a} = \begin{pmatrix} 4.1 & -0.1 & 2.2 \end{pmatrix}^\top$	$y_a = 1$	$p(c = y_a\|\mathbf{x}_a, T) = 100\%$
$\mathbf{x_b} = \begin{pmatrix} 6.1 & 0.4 & 1.3 \end{pmatrix}^\top$	$y_b = 2$	$p(c = y_b\|\mathbf{x}_b, T) = 66.6\%$

Table 1: Predicted classes for the given data points by the learned decision tree.

3 Problem 3

See the pdf pages attached.

4 Problem 4

Using the array function of the numpy library, the kNN algorithm can be implemented straightforwardly. The following source code shows the implemented kNN implementation.The predicted class labels for the vectors $\mathbf{x}_a$ and $\mathbf{x}_b$ equal to 2.

```python
import numpy as np

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))

class KNN():
    def __init__(self, k, data, labels):
        self.k = k
        self.data = data
        self.labels = np.array(labels)

    def predict(self, value):
        si = np.argsort([euclidean_distance(x, value) for x in
   self.data])
        labels = self.labels[si[0:self.k]]
        return self.labels[np.argmax(np.bincount(labels))]
```

Listing 2: Implementation of the kNN algorithm.

Data Point x	**Predicted Class** y
$\mathbf{x_a} = \begin{pmatrix} 4.1 & -0.1 & 2.2 \end{pmatrix}^\top$	$y_a = 2$
$\mathbf{x_b} = \begin{pmatrix} 6.1 & 0.4 & 1.3 \end{pmatrix}^\top$	$y_b = 2$

Table 2: Predicted classes for the given data points by kNN classifier.

5 Problem 5

Here the 3-NN classifier is used for regression. Both unweighted and weighted regression are compared. The weighted 3-NN regression uses the euclidean distance in order to determine the weights.

Data Point x	**Unweighted Regression Result** y	**Weighted Regression Result** y_w
$\mathbf{x_a} = \begin{pmatrix} 4.1 & -0.1 & 2.2 \end{pmatrix}^\top$	$y_a = 1.0$	$y_{w,a} = 0.56$
$\mathbf{x_b} = \begin{pmatrix} 6.1 & 0.4 & 1.3 \end{pmatrix}^\top$	$y_b = 1.33$	$y_{w,b} = 1.4$

Table 3: kNN for Regression. Predicted values for unweighted and weighted regression are presented.

6 Problem 6

Plotting the dataset reveals the distribution over the space. Figure 2 shows scatter plots of the data points as projections onto the three planes spanned by the principal axes. Obviously the feature spaces in the data set are completely different. Different scales and distributions are present. The Euclidean distance is like comparing apples and pears in this example. Another distance metric like the *Mahalanobis* distance promise a better results as this metric explicitly models different variances among different axes. Alternatively, the data can be preprocessed and scales to unit-variance per axis.

As the decision tree algorithm determines the decision threshold and dimensions based on measures like the entropy or the Gini-index, which are independent of the actual values in the feature space, decision trees are robust against different scales in the feature space.

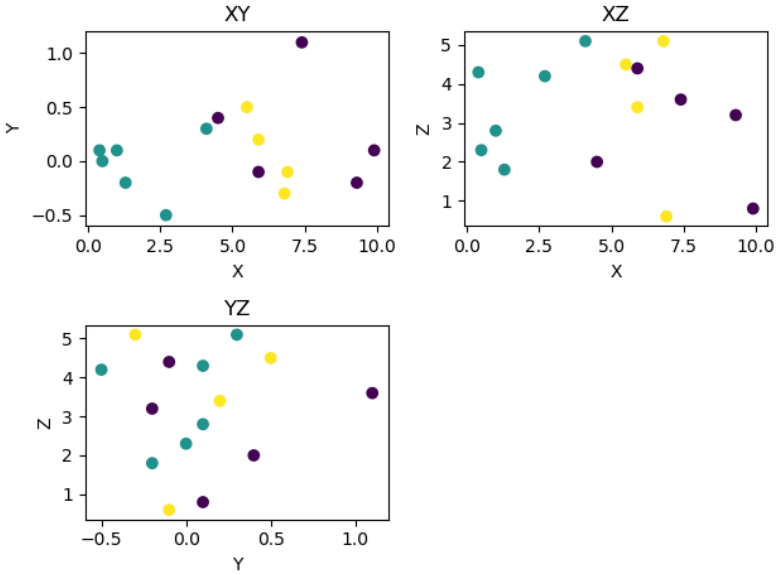


Figure 2: Plot of the dataset as projections onto the principal planes.

01_homework_knn

October 28, 2017

1 Programming assignment 1: k-Nearest Neighbors classification

```
In [3]: import numpy as np
        from sklearn import datasets, model_selection
        import matplotlib.pyplot as plt
        %matplotlib inline
```

1.1 Introduction

For those of you new to Python, there are lots of tutorials online, just pick whichever you like best :)

If you never worked with Numpy or Jupyter before, you can check out these guides * https://docs.scipy.org/doc/numpy-dev/user/quickstart.html * http://jupyter.readthedocs.io/en/latest/

1.2 Your task

In this notebook code to perform k-NN classification is provided. However, some functions are incomplete. Your task is to fill in the missing code and run the entire notebook.

In the beginning of every function there is docstring, which specifies the format of input and output. Write your code in a way that adheres to it. You may only use plain python and `numpy` functions (i.e. no scikit-learn classifiers).

Once you complete the assignments, export the entire notebook as PDF using nbconvert and attach it to your homework solutions. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

1.3 Load dataset

The iris data set (https://en.wikipedia.org/wiki/Iris_flower_data_set) is loaded and split into train and test parts by the function `load_dataset`.

```
In [4]: def load_dataset(split):
            """Load and split the dataset into training and test parts.

            Parameters
            ----------
            split : float in range (0, 1)
                Fraction of the data used for training.
```

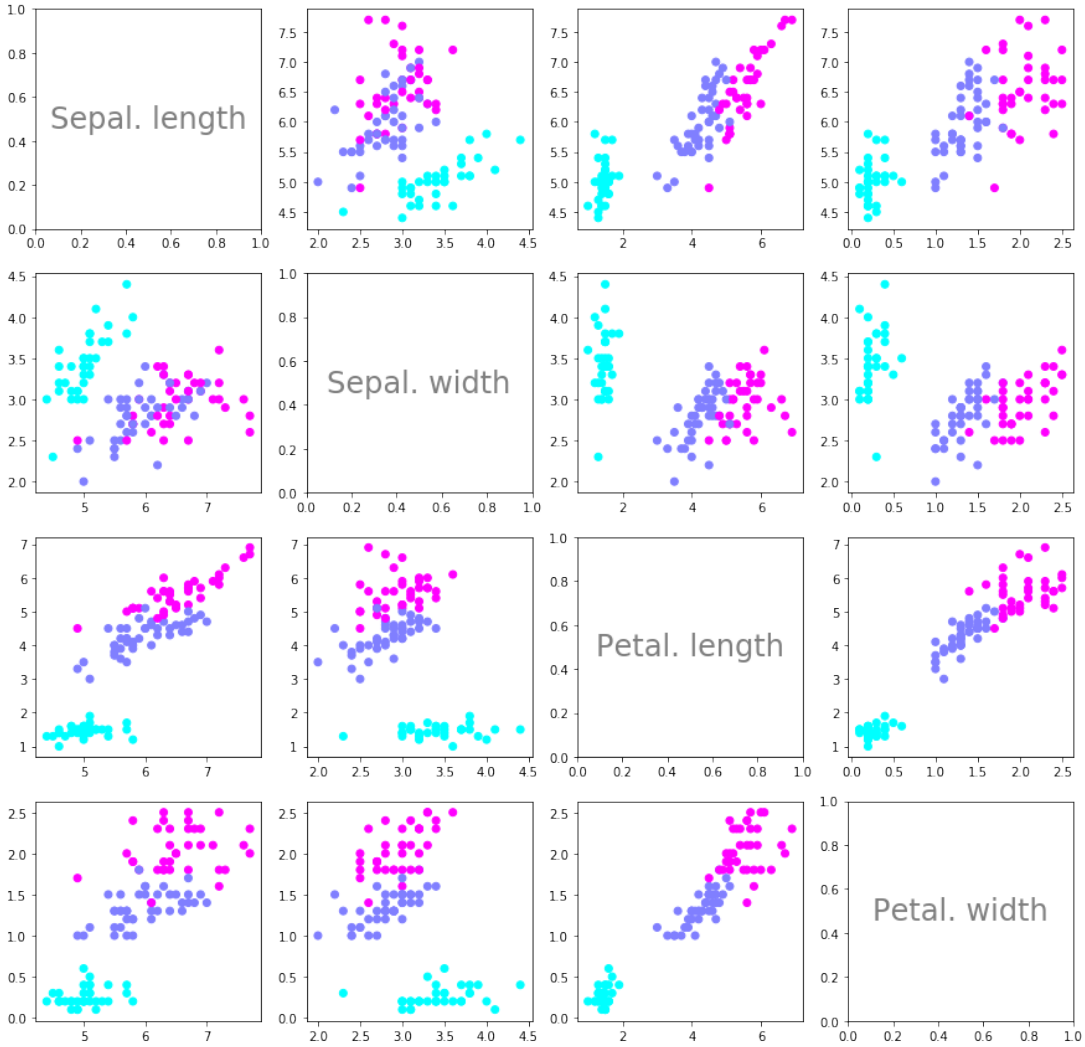
```
        Returns
        -------
        X_train : array, shape (N_train, 4)
            Training features.
        y_train : array, shape (N_train)
            Training labels.
        X_test : array, shape (N_test, 4)
            Test features.
        y_test : array, shape (N_test)
            Test labels.
        """
        dataset = datasets.load_iris()
        X, y = dataset['data'], dataset['target']
        X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, random_s
        return X_train, X_test, y_train, y_test
```

```
In [5]: # prepare data
        split = 0.75
        X_train, X_test, y_train, y_test = load_dataset(split)
```

1.4 Plot dataset

Since the data has 4 features, 16 scatterplots (4x4) are plotted showing the dependencies between each pair of features.

```
In [6]: f, axes = plt.subplots(4, 4,figsize=(15, 15))
        for i in range(4):
            for j in range(4):
                if j == 0 and i == 0:
                    axes[i,j].text(0.5, 0.5, 'Sepal. length', ha='center', va='center', size=2
                elif j == 1 and i == 1:
                    axes[i,j].text(0.5, 0.5, 'Sepal. width', ha='center', va='center', size=24
                elif j == 2 and i == 2:
                    axes[i,j].text(0.5, 0.5, 'Petal. length', ha='center', va='center', size=2
                elif j == 3 and i == 3:
                    axes[i,j].text(0.5, 0.5, 'Petal. width', ha='center', va='center', size=24
                else:
                    axes[i,j].scatter(X_train[:,j],X_train[:,i], c=y_train, cmap=plt.cm.cool)
```



1.5 Task 1: Euclidean distance

Compute Euclidean distance between two data points.

```
In [7]: def euclidean_distance(x1, x2):
            """Compute Euclidean distance between two data points.

            Parameters
            ----------
            x1 : array, shape (4)
                First data point.
            x2 : array, shape (4)
                Second data point.
```

```
    Returns
    -------
    distance : float
        Euclidean distance between x1 and x2.
    """
    return np.sqrt(np.sum((x1 - x2)**2))
```

1.6 Task 2: get k nearest neighbors' labels

Get the labels of the *k* nearest neighbors of the datapoint *x_new*.

```
In [8]: def get_neighbors_labels(X_train, y_train, x_new, k):
            """Get the labels of the k nearest neighbors of the datapoint x_new.

            Parameters
            ----------
            X_train : array, shape (N_train, 4)
                Training features.
            y_train : array, shape (N_train)
                Training labels.
            x_new : array, shape (4)
                Data point for which the neighbors have to be found.
            k : int
                Number of neighbors to return.

            Returns
            -------
            neighbors_labels : array, shape (k)
                Array containing the labels of the k nearest neighbors.
            """
            distances = [euclidean_distance(x, x_new) for x in X_train]
            sorted_indices = np.argsort(distances)
            result = list()
            for i in range(k):
                result.append(y_train[sorted_indices[i]])
            return result
```

1.7 Task 3: get the majority label

For the previously computed labels of the *k* nearest neighbors, compute the actual response. I.e. give back the class of the majority of nearest neighbors. In case of a tie, choose the "lowest" label (i.e. the order of tie resolutions is $0 > 1 > 2$).

```
In [9]: def get_response(neighbors_labels, num_classes=3):
            """Predict label given the set of neighbors.

            Parameters
            ----------
```

```
        neighbors_labels : array, shape (k)
            Array containing the labels of the k nearest neighbors.
        num_classes : int
            Number of classes in the dataset.

        Returns
        -------
        y : int
            Majority class among the neighbors.
        """
        class_votes = np.bincount(neighbors_labels)[0:num_classes]
        return np.argmax(class_votes)
```

1.8 Task 4: compute accuracy

Compute the accuracy of the generated predictions.

```
In [10]: def compute_accuracy(y_pred, y_test):
             """Compute accuracy of prediction.

             Parameters
             ----------
             y_pred : array, shape (N_test)
                 Predicted labels.
             y_test : array, shape (N_test)
                 True labels.
             """
             n_correct, n = 0, 0
             for pred, test in zip(y_pred, y_test):
                 n += 1
                 if pred == test:
                     n_correct += 1
             return n_correct / n
```

```
In [11]: # This function is given, nothing to do here.
         def predict(X_train, y_train, X_test, k):
             """Generate predictions for all points in the test set.

             Parameters
             ----------
             X_train : array, shape (N_train, 4)
                 Training features.
             y_train : array, shape (N_train)
                 Training labels.
             X_test : array, shape (N_test, 4)
                 Test features.
             k : int
                 Number of neighbors to consider.
```

```
    Returns
    -------
    y_pred : array, shape (N_test)
        Predictions for the test data.
    """
    y_pred = []
    for x_new in X_test:
        neighbors = get_neighbors_labels(X_train, y_train, x_new, k)
        y_pred.append(get_response(neighbors))
    return y_pred
```

1.9 Testing

Should output an accuracy of 0.9473684210526315.

```
In [13]: # prepare data
         split = 0.75
         X_train, X_test, y_train, y_test = load_dataset(split)
         print('Training set: {0} samples'.format(X_train.shape[0]))
         print('Test set: {0} samples'.format(X_test.shape[0]))

         # generate predictions
         k = 3
         y_pred = predict(X_train, y_train, X_test, k)
         accuracy = compute_accuracy(y_pred, y_test)
         print('Accuracy = {0}'.format(accuracy))
```

```
Training set: 112 samples
Test set: 38 samples
Accuracy = 0.9473684210526315
```

Machine Learning Worksheet 2

Probability Theory

1 Basic Probability

Problem 1: A secret government agency has developed a scanner which determines whether a person is a terrorist. The scanner is fairly reliable; 95% of all scanned terrorists are identified as terrorists, and 95% of all upstanding citizens are identified as such. An informant tells the agency that exactly one passenger of 100 aboard an airplane in which you are seated is a terrorist. The agency decide to scan each passenger and the shifty looking man sitting next to you is tested as "TERRORIST". What are the chances that this man *is* a terrorist? Show your work!

Problem 2: A fair coin is tossed twice. Whenever it turns up heads, a red ball is placed into a box, otherwise a white ball. Afterwards, balls are drawn from the box three times in succession (placing the drawn ball back into the box every time). It is found that on all three occasions a red ball is drawn. What is the probability that both balls in the box are red? Show your work!

Problem 3: A fair coin is flipped until heads shows up for the first time. What is the expected number of tails T and the expected number of heads H in any one run of this experiment? Show your work.

Hint: While there is a very short solution to this problem for people with a good intuition, the rest of us might need to look at the geometric series *and its properties. You may use them without proof.*

Problem 4: Calculate mean and variance of a uniform random variable X on the interval $[a, b], a < b$ with probability density function

$$p(x) = \begin{cases} \frac{1}{b-a} & a \le x \le b, \\ 0 & \text{elsewhere.} \end{cases}$$

Problem 5: Let X and Y be random variables with joint density $p(x, y)$. Prove the *tower properties*,

$$E[X] = E_Y[E_{X|Y}[X]],$$
$$\mathrm{Var}[X] = E_Y[\mathrm{Var}_{X|Y}[X]] + \mathrm{Var}_Y[E_{X|Y}[X]].$$

$E_{X|Y}[X]$ and $\mathrm{Var}_{X|Y}[X]$ denote the expectation and variance of X under the conditional density $p(x \mid y)$.

2 Probability Inequalities

Inequalities are useful for bounding quantities that might otherwise be hard to compute. A famous example is the Markov inequality

$$p(X > c) \leq \frac{E[X]}{c}$$

for a *non-negative* random variable X and a constant $c > 0$. From it, it is relatively easy to prove the Chebyshev inequality

$$p(|X - E[X]| > c) \leq \frac{\mathrm{Var}(X)}{c^2}$$

for arbitrary X with finite variance.

With the help of Chebyshev's inequality, one can prove (a weak version of) the *law of large numbers*, which roughly states that the empirical mean of n i.i.d. random variables X_i converges to the true mean for $n \to \infty$. More formally, for any $\epsilon > 0$

$$p\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - E[X_i]\right| > \epsilon\right) \to 0 \qquad \text{as } n \to \infty. \tag{1}$$

Problem 6: Prove eq. (1). You may assume that the X_i have finite variance $\mathrm{Var}[X_i]$. You may further use Markov's and Chebyshev's inequalities without proof.

(We highly recommend to practise your "proof skills" on them, though. The proofs are technical, but very short.)

Homework 02 - Probability Theory

IN2064 Machine Learning

Arne Sachtler - *Registration Number: 03692662*

November 5, 2017

1 Basic Probability

1.1 Problem 1

Introducing event variables as

$$T : \text{person is a terrorist} \tag{1}$$

and

$$S : \text{person is assessed as terrorist} \tag{2}$$

leads to a formal specification of the knowledge provided in the textual description as

$$P(S|T) = 95\% \quad , \quad P(\bar{S}|\bar{T}) = 95\% \quad \text{and} \quad P(T) = 1\%. \tag{3}$$

In order to determine the probability that the person next to me actually is a terrorist, the conditional probability $P(T|S)$ is required. Applying Bayes' rule yields

$$\begin{aligned} P(T|S) &= \frac{P(S|T)P(T)}{P(S)} && (4)\\ &= \frac{P(S|T)P(T)}{\sum_{A\in\{T,\bar{T}\}} P(A)P(S|A)} && (5)\\ &= \frac{95\% \cdot 1\%}{1\% \cdot 95\% + 99\% \cdot 5\%} && (6)\\ &\approx 16.1\% && (7) \end{aligned}$$

Lucky me, the person next to me is a terrorist by only 16%.

1.2 Problem 2

Let the random variable C be the amount of red balls placed into the box and D the amount of red balls drawn from the box. C and D are Binomial distributed as follows

$$\begin{aligned} C &\sim Bin\left(2, \frac{1}{2}\right) && (8)\\ D &\sim Bin\left(3, \mu_r\right) , && (9) \end{aligned}$$

where the parameter μ_r denotes the probability of drawing a red ball from the box. This parameter is dependent on the amount of red balls in the box. Formally the required conditional probability can be expressed in terms of those random variables as $p(C = 2|D = 3)$. Again, Bayes' rule can be applied in order to compute the final conditional probability. The law of marginal probability is used and the parameter μ_r is adopted for each case. In the following equations let $B(k|N, \mu)$ be the short form of $Bin(k|N, \mu)$.

$$
\begin{aligned}
p(C = 2|D = 3) &= \frac{p(D = 3|C = 2)\, p(C = 2)}{\sum_{k=0}^{2} p(D = 3|C = k)\, p(C = k)} && (10)\\
&= \frac{B(3|3, 1)\, B(2|2, 0.5)}{B(3|3, 0)\, B(0|2, 0.5) + B(3|3, 0.5)\, B(1|2, 0.5) + B(3|3, 1)\, B(2|2, 0.5)} && (11)\\
&= \frac{1 \cdot \frac{1}{4}}{0 \cdot \frac{1}{4} + \frac{1}{8} \cdot \frac{1}{2} + 1 \cdot \frac{1}{4}} && (12)\\
&= \frac{4}{5} = 80\%\,. && (13)\\
& && (14)
\end{aligned}
$$

Under the observation of drawing three times a red ball from the box, the probability that initially two of the two balls placed in the box were red is 80%.

1.3 Problem 3

Let N be the random variable modelling the trial until at first the head appears. The probability, that a head appears after the first trial is p. Afterwards, the probability that the first head appears after the second trial is $p(1 - p)$. Therefore, the probability for the first head appearing after the n-th trial is

$$p(N = n) = p(1 - p)^{n-1}\,. \tag{15}$$

In this case, for the fair coin, p equal $\frac{1}{2}$ and $p(N = n)$ can be simplified to

$$p(N = n) = \left(\frac{1}{2}\right)^n = \frac{1}{2^n}\,. \tag{16}$$

The function evaluating the number of trials after n trial is simply the identity:

$$g(n) = n\,. \tag{17}$$

Finally, the expected value of trial $E\left[g(n)\right]$ can be computed by

$$E\left[g(n)\right] = \sum_{k=1}^{\infty} \frac{n}{2^n} = 2 \tag{18}$$

The expected number of heads is of course one, a each run of the experiment terminates after the first heads appeared:

$$E[H] = 1\,. \tag{19}$$

Provided the first heads appeared after n trials, $n-1$ tails have been observed in advance. Therefore the expected number of tails is expressed by the following sum

$$\begin{aligned} E[T] &= \sum_{k=1}^{\infty} \frac{n-1}{2^n} && (20)\\ &= \sum_{k=1}^{\infty} \frac{n}{2^n} - \sum_{k=1}^{\infty} \frac{1}{2^n} && (21)\\ &= 2 - 1 = 1\,. && (22) \end{aligned}$$

1.4 Problem 4

For the expected value, the fact is used that $p(x) = 0$ for $x < a$ and $x > b$. First the expected value:

$$\begin{aligned} E[X] &= \int_{-\infty}^{\infty} sp(s)ds = \frac{1}{b-a}\int_{a}^{b} sds && (23)\\ &= \frac{1}{b-a}\left(\frac{1}{2}b^2 - \frac{1}{2}a^2\right) && (24)\\ &= \frac{(b+a)(b-a)}{2(b-a)} && (25)\\ &= \frac{b+a}{2}\,. && (26) \end{aligned}$$

And for the variance:

$$\begin{aligned} Var\left[X\right] &= E\left[X^2\right] - E\left[X\right]^2 && (27)\\ &= \int_{-\infty}^{\infty} s^2 p(s) ds - \left(\int_{-\infty}^{\infty} sp(s) ds \right)^2 && (28)\\ &= \frac{1}{b-a} \int_a^b s^2 ds - \frac{(a+b)^2}{4} && (29)\\ &= \frac{1}{b-a} \left(\frac{1}{3} b^3 - \frac{1}{3} a^3 \right) - \frac{(a+b)^2}{4} && (30)\\ &= \frac{b^3 - a^3}{3(b-a)} - \frac{(a+b)^2}{4} && (31)\\ &= \frac{4b^3 - 4a^3 - \left(3a^2 + 6ab + 3b^2\right)(b-a)}{12(b-a)} && (32)\\ &= \frac{b^3 - 3ab^2 + 3a^2 b - a^3}{12(b-a)} && (33)\\ &= \frac{(b-a)^2}{12} && (34) \end{aligned}$$

1.5 Problem 5

First the expected value. Using the law of marginal probability

$$\int p(x|y)p(y)dy = p(x)\,, \tag{35}$$

the proof can be constructed straightforwardly

$$\begin{aligned} E\left[X\right] &= E_Y\left[E_{X|Y}\left[X\right]\right] && (36)\\ &= \int E_{X|Y}\left[X\right] p(y) dy && (37)\\ &= \iint xp(x|y)p(y)\, dx\, dy && (38)\\ &= \int xp(x) dx && (39)\\ &= E\left[X\right]\,. && (40) \end{aligned}$$

Using the equalities

$$E_Y\left[Var_{X|Y}\left[X\right]\right] = \iint x^2 p(x|y)p(y) dx dy - \int \left(\int xp(x|y) dx \right)^2 p(y) dy \tag{41}$$

and

$$Var_Y\left[E_{X|Y}\left[X\right]\right] = \int\left(\int xp(x|y)dx\right)^2 p(y)dy - \left(\iint x(p(x|y)p(y)dxdy\right)^2 \tag{42}$$

the tower probability for the variance is proven

$$\begin{aligned}
& E_Y\left[Var_{X|Y}\left[X\right]\right] + Var_Y\left[E_{X|Y}\left[X\right]\right] && (43)\\
= \quad & \iint x^2 p(x|y)p(y)dxdy - \int\left(\int xp(x|y)dx\right)^2 p(y)dy && (44)\\
+ \quad & \int\left(\int xp(x|y)dx\right)^2 p(y)dy - \left(\iint x(p(x|y)p(y)dxdy\right)^2 && (45)\\
= \quad & \iint x^2 p(x|y)p(y)dxdy - \left(\iint x(p(x|y)p(y)dxdy\right)^2 && (46)\\
= \quad & \int x^2 p(x)dx - \left(\int xp(x)dx\right)^2 && (47)\\
= \quad & E\left[X^2\right] - E\left[X\right]^2 && (48)\\
= \quad & Var\left[X\right]. && (49)
\end{aligned}$$

■

2 Probability Inequalities

2.1 Problem 6

To prove:

$$\lim_{n\to\infty} p\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - E\left[X_i\right]\right| > \epsilon\right) = 0\,. \tag{50}$$

Using $X = \sum X_i$ the term can be bounded by Chebyshev's inequality:

$$\begin{aligned}
& p\left(\left|\tfrac{1}{n}\sum_{i=1}^{n} X_i - E\left[X_i\right]\right| > \epsilon\right) && (51)\\
= \quad & p\left(\left|X - E\left[X\right]\right| > n\epsilon\right) && (52)\\
\leq \quad & \tfrac{Var[X]}{n^2\epsilon^2}\,. && (53)
\end{aligned}$$

For $n \to \infty$ the term $\frac{Var[X]}{n^2\epsilon^2}$ goes to zero. ■

Machine Learning Worksheet 3

Parameter Inference

1 Optimizing Likelihoods: Monotonic Transforms

Usually one considers the *log likelihood*, $\log p(x_1, \ldots, x_n \mid \theta)$. The next problems justify this.

In the lecture, we encountered the likelihood maximization problem

$$\arg\max_{\theta \in [0,1]} \theta^t (1-\theta)^h,$$

where t and h denoted the number of tails and heads in a sequence of coin tosses, respectively.

Problem 1: Compute the first and second derivative of this likelihood w.r.t. θ. Then compute first and second derivative of the log likelihood $\log \theta^t (1-\theta)^h$.

Problem 2: Show that every local maximum of $\log f(\theta)$ is also a local maximum of the differentiable, positive function $f(\theta)$. Considering this and the previous exercise, what is your conclusion?

2 Properties of MLE and MAP

Problem 3: Show that θ_{MLE} can be interpreted as a special case of θ_{MAP} in the sense that there always exists a prior $p(\theta)$ such that $\theta_{\text{MLE}} = \theta_{\text{MAP}}$.

Problem 4: Consider a Bernoulli random variable X and suppose we have observed m occurrences of $X = 1$ and l occurrences of $X = 0$ in a sequence of $N = m + l$ Bernoulli experiments. We are only interested in the number of occurrences of $X = 1$—we will model this with a Binomial distribution with parameter θ. A prior distribution for θ is given by the Beta distribution with parameters a, b. Show that the posterior *mean* value $E[\theta \mid \mathcal{D}]$ (not the MAP estimate) of θ lies between the prior mean of θ and the maximum likelihood estimate for θ.

To do this, show that the posterior mean can be written as λ times the prior mean plus $(1-\lambda)$ times the maximum likelihood estimate, with $0 \leq \lambda \leq 1$. This illustrates the concept of the posterior mean being a compromise between the prior distribution and the maximum likelihood solution.

The probability mass function of the Binomial distribution for some $m \in \{0, 1, \ldots, N\}$ is

$$p(x = m \mid N, \theta) = \binom{N}{m} \theta^m (1-\theta)^{N-m}.$$

Hint: Identify the posterior distribution. You may then look up the mean rather than computing it.

3 Poisson Distribution

Problem 5: Let X be Poisson distributed. Again, for n i.i.d. samples from X, determine the maximum likelihood estimate for λ.

In class we also talked about avoiding overfitting of parameters via *prior* information. Compute the posterior distribution over λ, assuming a Gamma(α, β) prior for it. Compute the MAP for λ under this prior. Show your work.

Homework 03 - Paramter Inference

IN2064 Machine Learning

Arne Sachtler - *Registration Number: 03692662*

12th November 2017

1 Optimizing Likelihoods: Monotonic Transforms

1.1 Problem 1

First derivative of the likelihood:

$$\frac{d}{d\theta}\theta^t (1-\theta)^h = t\theta^{t-1} (1-\theta)^h - \theta^h h (1-\theta)^{h-1} \,. \tag{1}$$

and the second derivative:

$$\begin{aligned} \frac{d^2}{d\theta^2}\theta^t (1-\theta)^h &= t(t-1)\theta^{t-2}(1-\theta)^h - ht\theta^{t-1}(1-\theta)^{h-1} &(2)\\ &- ht\theta^{t-1}(1-\theta)^{h-1} + h(h-1)\theta^t(1-\theta)^{h-2}\,. &(3)\end{aligned}$$

Compared to the derivatives of the log likelihood

$$\frac{d}{d\theta}\ln\left(\theta^t (1-\theta)^h\right) = \frac{t}{\theta} - \frac{h}{\theta}\,, \tag{4}$$

and

$$\frac{d^2}{d\theta^2}\ln\left(\theta^t (1-\theta)^h\right) = -\frac{t}{\theta^2} + \frac{t}{\theta^2}\,. \tag{5}$$

Due to the sums in the log likelihood the terms are much simpler.

1.2 Problem 2

Let x_f be a maximum of $f(x)$, that is

$$f'(x_f) = 0 \quad \wedge \quad f''(x_f) < 0\,. \tag{6}$$

Further, let $lf(x) = \log f(x)$ be the logarithmic function and let there be a (hypothetical) $\tilde{x}$ that is a maximum of $f(x)$ and no maximum $\log f(x)$. More formally

$$\exists \tilde{x} \in \mathbb{R} : f'(\tilde{x}) = 0 \quad \wedge \quad f''(\tilde{x}) < 0 \quad \wedge \quad lf'(\tilde{x}) \neq 0 \tag{7}$$

Then it follows that

$$lf'(\tilde{x}) = (\log f)'(\tilde{x}) = f'(\tilde{x})\frac{1}{f(\tilde{x})} \neq 0 \lightning, \tag{8}$$

which is a contradiction as $f'(\tilde{x}) = 0$. It is proven by contraction that

$$\forall x_f \in \mathbb{R} : f'(\tilde{x}) = 0 \quad \Leftrightarrow \quad lf'(\tilde{x}) = 0\,. \tag{9}$$

Consequently every maximum of f is a maximum of $\log f$. ■

2 Properties of MLE and MAP

2.1 Problem 3

Show

$$\exists p(\theta) : \theta_{MAP} = \theta_{MLE} \tag{10}$$

We know that

$$\theta_{MLE} = \arg\max_{\theta} p(D|\theta) \tag{11}$$

and

$$\theta_{MAP} = \arg\max_{\theta} \frac{p(D|\theta)p(\theta)}{p(D)} \tag{12}$$

Let $p(\theta) = p_0, p_0 \in \mathbb{R}\setminus\{0\}$ be a constant distribution. Thus for $p(\theta) > 0$ it holds that

$$\theta_{MLE} = \arg\max_{\theta} p(D|\theta) = \arg\max_{\theta} \frac{p(D|\theta)p(\theta)}{p(D)} = \theta_{MAP}\,, \tag{13}$$

as $p(\theta) = p_0$ is constant and $p(D)$ does not depend on θ.

For a constant prior distribution the maximum likelihood estimate equals the maximum a posteriori estimate.

2.2 Problem 4

The prior is a Beta distribution with parameters a and b. We know for the prior

$$p(\theta) = Beta(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}\theta^{a-1}(1-\theta)^{b-1}\,. \tag{14}$$

Further we know that the likelihood distribution is Binomial with parameters N, m and θ:

$$p(x = m|N,\theta) = \binom{N}{m}\theta^m(1-\theta)^{N-m}\,. \tag{15}$$

Let

$$\eta = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}\binom{N}{m}\theta^m\frac{1}{p(D)}\,. \tag{16}$$

Obviously, the normalizing factor η does not depend on θ and the resulting posterior is computed by the product of prior and likelihood

$$p(\theta|D) = \eta\theta^{m+a-1}(1-\theta)^{N+b-m-1}\,, \tag{17}$$

which is a Beta-distribution again. Consequently the resulting posterior is $Beta(m + a, N + b - m) = Beta(m + a, l + b)$ distributed. The mean of a $Beta(a, b)$ distribution is

$$X \sim Beta(a,b) \quad , \quad E[X] = \frac{a}{a+b}\,. \tag{18}$$

For the prior mean we get

$$E[\theta] = \frac{a}{a+b}\,, \tag{19}$$

for the posterior mean it follows that

$$E[\theta|D] = \frac{m+a}{m+l+a+b}\,, \tag{20}$$

and finally the maximum likelihood estimate is

$$\theta_{MLE} = \frac{m}{m+l}\,. \tag{21}$$

Still to show:

$$\exists\lambda \in [0,1] : E[\theta|D] = \lambda E[\theta] + (1-\lambda)\theta_{MLE}\,. \tag{22}$$

Combining this with the results for the single estimates yields

$$\frac{m+a}{m+l+a+b} = \lambda\frac{a}{a+b} + (1-\lambda)\frac{m}{m+l} \tag{23}$$

solving for λ yields

$$\lambda = \frac{b+a}{N+a+b}\,. \tag{24}$$

As all number are positive is follows that $0 < \lambda < 1$. ■.

3 Poison Distribution

3.1 Problem 5

Let $X \sim Poi(\lambda)$ be Poisson distributed. An experiments with n i.i.d. samples was performed and k hits were observed. It follows that the likelihood function is

$$p(\lambda|n,k) = \frac{\lambda^k}{k!}e^{-\lambda}\,. \tag{25}$$

The maximum likelihood estimate is computed setting the first derivative with respect to λ to zero. Therefore the log likelihood is computed

$$\log p(\lambda|n,k) = k\log\lambda - \log k! - \lambda\,, \tag{26}$$

and the first derivative is set to zero

$$0 \stackrel{!}{=} \frac{d}{d\lambda} \log p(\lambda|n,k) = \frac{k}{\lambda} - 1\,. \tag{27}$$

And it follows that $\lambda_{MLE} = k$.

For the MAP a $Gamma(\alpha, \beta)$ distributed prior is used. It follows that λ is distributed according to

$$p(\lambda|\alpha,\beta) = \frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)} \tag{28}$$

For the posterior distribution we get

$$p(\lambda|D) = \frac{\frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)} \frac{\lambda^k}{k!} e^{-\lambda}}{p(D)} = \eta \lambda^{k+\alpha-1} e^{-(\beta+1)\lambda}\,. \tag{29}$$

Thus the posterior is $Gamma(k+\alpha, \beta+1)$ distributed. An the maximum a posteriori estimate becomes

$$\lambda_{MAP} = \frac{k+\alpha-1}{\beta+1}\,. \tag{30}$$

Machine Learning Worksheet 04

Linear Regression

---

1 Least squares regression

Problem 1: Load the notebook `04_homework_linear_regression.ipynb` from Piazza. Fill in the missing code and run the notebook. Convert the evaluated notebook to pdf and add it to the printout of your homework.

Note: We suggest that you use Anaconda for installing Python and Jupyter, as well as for managing packages. We recommend that you use Python 3.

For more information on Jupyter notebooks and how to convert them to other formats, consult the Jupyter documentation and nbconvert documentation.

Problem 2: Let's assume we have a dataset where each datapoint, $(\boldsymbol{x}_i, y_i)$ is weighted by a scalar factor which we will call t_i. We will assume that $t_i > 0$ for all i. This makes the sum of squares error function look like the following:

$$E_{\text{weighted}}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N} t_i \left[\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i) - y_i\right]^2$$

Find the equation for the value of $\boldsymbol{w}$ that minimizes this error function.

Furthermore, explain how this weighting factor, t_i, can be interpreted in terms of
1) the variance of the noise on the data and
2) data points for which there are exact copies in the dataset.

2 Ridge regression

Problem 3: Show that the following holds: The ridge regression estimates can be obtained by ordinary least squares regression on an augmented dataset: Augment the design matrix $\boldsymbol{\Phi} \in \mathbb{R}^{N\times M}$ with M additional rows $\sqrt{\lambda}\boldsymbol{I}_{M\times M}$ and augment $\boldsymbol{y}$ with M zeros.

3 Bayesian linear regression

In the lecture we made the assumption that we already knew the precision (inverse variance) for our Gaussian distributions. What about when we don't know the precision and we need to put a prior on that as well as our Gaussian prior that we already have on the weights of the model?

Upload a single PDF file with your solution to Moodle by 20.11.2017, 9:59AM CET. The solution must be typeset (using Latex, Word, etc.). Handwritten solutions are not accepted.

Problem 4: It turns out that the conjugate prior for the situation when we have an unknown mean and unknown precision is a normal-gamma distribution (See section 2.3.6 in Bishop). This is also true when we have a conditional Gaussian distribution of the linear regression model. This means that if our likelihood is as follows:

$$p(\boldsymbol{y} \mid \boldsymbol{\Phi}, \boldsymbol{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(y_i \mid \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i), \beta^{-1})$$

Then the conjugate prior for both $\boldsymbol{w}$ and β is

$$p(\boldsymbol{w}, \beta) = \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{m}_0, \beta^{-1}\boldsymbol{S}_0)\text{Gamma}(\beta \mid a_0, b_0)$$

Show that the posterior distribution takes the same form as the prior, i.e.

$$p(\boldsymbol{w}, \beta \mid \mathcal{D}) = \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{m}_N, \beta^{-1}\boldsymbol{S}_N)\text{Gamma}(\beta \mid a_N, b_N)$$

Also be sure to give the expressions for $\boldsymbol{m}_N$, $\boldsymbol{S}_N$, a_N, and b_N.

Homework 04 - Linear Regression

IN2064 Machine Learning

Arne Sachtler - *Registration Number: 03692662*

19th November 2017

1. Least Squares Regression

1.1. Problem 1

See the generate pages in appendix A.

1.2. Problem 2

Given loss function:

$$E_{weighted}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} t_i \left[\mathbf{w}^\top \phi(\mathbf{x}_i) - y_i \right]^2 . \tag{1}$$

Introduction of a diagonal matrix $\mathbf{T}$ with the weights t_i on its diagonal enables one to rewrite the given loss function in full matrix notation with the design matrix $\mathbf{\Phi}$ and the full vector of target values $\mathbf{y}$. This yields

$$\begin{aligned} E_w(\mathbf{w}) &= \frac{1}{2} \left(\mathbf{\Phi w} - \mathbf{y} \right)^\top \mathbf{T} \left(\mathbf{\Phi w} - \mathbf{y} \right) && (2) \\ &= \frac{1}{2} \left(\mathbf{w}^\top \mathbf{\Phi}^\top \mathbf{T} \mathbf{\Phi w} - 2 \left(\mathbf{y}^\top \mathbf{T} \mathbf{\Phi w} \right) + \mathbf{y}^\top \mathbf{T y} \right) , && (3) \end{aligned}$$

where

$$\mathbf{T} = \begin{pmatrix} t_1 & & \dots & 0 \\ & t_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \dots & & t_n \end{pmatrix} . \tag{4}$$

Computing the derivative with respect to $\mathbf{w}$ yields

$$\frac{\partial E_w}{\partial \mathbf{w}} = \left(\mathbf{\Phi}^\top \mathbf{T} \mathbf{\Phi w} - \mathbf{y}^\top \mathbf{T} \mathbf{\Phi} \right) . \tag{5}$$

And consequently is follows that

$$\hat{\mathbf{w}} = \left(\mathbf{\Phi}^\top \mathbf{T} \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^\top \mathbf{T} \mathbf{y} \,. \tag{6}$$

The parameters t_i act like a precision factor in the expression. Inspecting equation 2 reveals that the matrix of weights $\mathbf{T}$ acts like the inverse of a covariance matrix in a multinomial normal distribution. We can see that $\mathbf{T}$ is comparable to $\mathbf{\Sigma}^{-1}$.

2. Ridge Regression

2.1. Problem 3

Ordinary linear least squares finds the minimum of the loss function

$$E = \frac{1}{2} \left(\mathbf{\Phi}\mathbf{w} - \mathbf{y}\right)^\top \left(\mathbf{\Phi}\mathbf{w} - \mathbf{y}\right) \,. \tag{7}$$

Using the augmented data set matrices as

$$\mathbf{\Psi} = \begin{pmatrix} \mathbf{\Phi} \\ \sqrt{\lambda}\mathbf{I}_M \end{pmatrix} \quad \text{and} \quad \mathbf{z} = \begin{pmatrix} \mathbf{y} \\ \mathbf{0}_M \end{pmatrix} \tag{8}$$

and evaluated the loss function for ordinary least squared with these yields

$$\begin{aligned} E &= \frac{1}{2} \left(\mathbf{\Psi}\mathbf{w} - \mathbf{z}\right)^\top \left(\mathbf{\Psi}\mathbf{w} - \mathbf{z}\right) && (9) \\ &= \frac{1}{2} \left(\begin{pmatrix} \mathbf{\Phi}\mathbf{w} \\ \sqrt{\lambda}\mathbf{I}_M\mathbf{w} \end{pmatrix} - \begin{pmatrix} \mathbf{y} \\ \mathbf{0}_M \end{pmatrix}\right)^\top \left(\begin{pmatrix} \mathbf{\Phi}\mathbf{w} \\ \sqrt{\lambda}\mathbf{I}_M\mathbf{w} \end{pmatrix} - \begin{pmatrix} \mathbf{y} \\ \mathbf{0}_M \end{pmatrix}\right) && (10) \\ &= \frac{1}{2} \begin{pmatrix} \left(\mathbf{\Phi}\mathbf{w} - \mathbf{y}\right)^\top & \sqrt{\lambda}\mathbf{w}^\top \end{pmatrix} \begin{pmatrix} \left(\mathbf{\Phi}\mathbf{w} - \mathbf{y}\right)^\top \\ \sqrt{\lambda}\mathbf{w}^\top \end{pmatrix} && (11) \\ &= \frac{1}{2} \left(\mathbf{\Phi}\mathbf{w} - \mathbf{y}\right)^\top \left(\mathbf{\Phi}\mathbf{w} - \mathbf{y}\right) + \frac{1}{2}\lambda \mathbf{w}^\top \mathbf{w} \,, && (12) \end{aligned}$$

which is exactly the same loss function as that solved by ridge regression. ■

A. Notebook for the Linear Regression Task

Programming assignment 4: Linear regression

In [1]:
```
import numpy as np

from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
```

Your task

In this notebook code skeleton for performing linear regression is given. Your task is to complete the functions where required. You are only allowed to use built-in Python functions, as well as any numpy functions. No other libraries / imports are allowed.

Load and preprocess the data

I this assignment we will work with the Boston Housing Dataset. The data consists of 506 samples. Each sample represents a district in the city of Boston and has 13 features, such as crime rate or taxation level. The regression target is the median house price in the given district (in $1000's).

More details can be found here: http://lib.stat.cmu.edu/datasets/boston (http://lib.stat.cmu.edu/datasets/boston)

In [8]:
```
X , y = load_boston(return_X_y=True)

# Add a vector of ones to the data matrix to absorb the bias term
# (Recall slide #7 from the lecture)
X = np.hstack([np.ones([X.shape[0], 1]), X])
# From now on, D refers to the number of features in the AUGMENTED da
taset (i.e. including the dummy '1' feature for the absorbed bias ter
m)

# Split into train and test
test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=t
est_size)
```

Task 1: Fit standard linear regression

In [27]:
```
def fit_least_squares(X, y):
    """Fit ordinary least squares model to the data.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Regression targets.

    Returns
    -------
    w : array, shape [D]
        Optimal regression coefficients (w[0] is the bias term).

    """
    w = np.linalg.lstsq(X, y)[0]
    return w
```

Task 2: Fit ridge regression

In [28]:
```
def fit_ridge(X, y, reg_strength):
    """Fit ridge regression model to the data.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Regression targets.
    reg_strength : float
        L2 regularization strength (denoted by lambda in the lecture)

    Returns
    -------
    w : array, shape [D]
        Optimal regression coefficients (w[0] is the bias term).

    """
    w = np.linalg.inv(X.T.dot(X) + reg_strength*np.eye(X.shape[1])).d
ot(X.T).dot(y)
    return w
```

Task 3: Generate predictions for new data

In [21]:
```
def predict_linear_model(X, w):
    """Generate predictions for the given samples.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    w : array, shape [D]
        Regression coefficients.

    Returns
    -------
    y_pred : array, shape [N]
        Predicted regression targets for the input data.

    """
    return w.dot(X.T)
```

Task 4: Mean squared error

In [30]:
```
def mean_squared_error(y_true, y_pred):
    """Compute mean squared error between true and predicted regressi
on targets.

    Reference: `https://en.wikipedia.org/wiki/Mean_squared_error`

    Parameters
    ----------
    y_true : array
        True regression targets.
    y_pred : array
        Predicted regression targets.

    Returns
    -------
    mse : float
        Mean squared error.

    """
    return np.sum((y_true - y_pred)**2)/y_true.size
```

Compare the two models

The reference implementation produces

- MSE for Least squares ≈ **23.98**
- MSE for Ridge regression ≈ **21.05**

You results might be slightly (i.e. $\pm 1\%$) different from the reference soultion due to numerical reasons.

In [31]:
```
# Load the data
np.random.seed(1234)
X , y = load_boston(return_X_y=True)
X = np.hstack([np.ones([X.shape[0], 1]), X])
test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=t
est_size)

# Ordinary least squares regression
w_ls = fit_least_squares(X_train, y_train)
y_pred_ls = predict_linear_model(X_test, w_ls)
mse_ls = mean_squared_error(y_test, y_pred_ls)
print('MSE for Least squares = {0}'.format(mse_ls))

# Ridge regression
reg_strength = 1
w_ridge = fit_ridge(X_train, y_train, reg_strength)
y_pred_ridge = predict_linear_model(X_test, w_ridge)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
print('MSE for Ridge regression = {0}'.format(mse_ridge))
```

```
MSE for Least squares = 23.984307611784356
MSE for Ridge regression = 21.051487033772197
```

Machine Learning Worksheet 05

Linear Classification

1 Linear separability

Problem 1: Given a set of data points $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^N$, we can define the *convex hull* co$\mathcal{X}$ to be the set of all points $\boldsymbol{x}$ given by

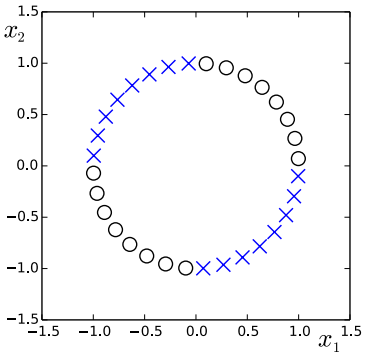
$$\text{co}\mathcal{X} = \{\boldsymbol{x} : \boldsymbol{x} = \sum_i \alpha_i \boldsymbol{x}_i, \alpha_i \geq 0, \sum_i \alpha_i = 1\}$$

Consider a second set of points $\mathcal{Y} = \{\boldsymbol{y}_j\}_{j=1}^M$ together with its corresponding convex hull. By definition, the two sets of points will be linearly separable if there exists a vector $\boldsymbol{w}$ and a scalar w_0 such that $\boldsymbol{w}^T\boldsymbol{x}_i + w_0 > 0$ for all $\boldsymbol{x}_i \in \mathcal{X}$, and $\boldsymbol{w}^T\boldsymbol{y}_j + w_0 < 0$ for all $\boldsymbol{y}_j \in \mathcal{Y}$. Show that if their convex hulls intersect, the two sets of points cannot be linearly separable.

Problem 2: Show that for a linearly separable data set, the maximum likelihood solution for the logistic regression model is obtained by finding a vector $\boldsymbol{w}$ whose decision boundary $\boldsymbol{w}^T\boldsymbol{x} = 0$ separates the classes and then taking the magnitude of $\boldsymbol{w}$ to infinity. Assume that $\boldsymbol{w}$ contains the bias term.

How can we prevent this?

Problem 3: Which basis function $\phi(x_1, x_2)$ makes the data in the example below linearly separable (crosses in one class, circles in the other)?



2 Basis functions

Problem 4: The decision boundary for a linear classifier on two-dimensional data crosses axis x_1 at 2 and x_2 at 5. Write down the general form of this linear classifier model with a bias term (how many parameters do you need, given the dimensions?) and calculate possible coefficients (parameters).

Upload a single PDF file with your solution to Moodle by 27.11.2017, 9:59am CET. The solution must be typeset (using Latex, Word, etc.). Handwritten solutions are not accepted.

Homework 05 - Linear Classification

IN2064 Machine Learning

Arne Sachtler - *Registration Number: 03692662*

November 27, 2017

1 Linear Separability

1.1 Problem 1

The given convex hulls intersect, therefore we have

$$\exists \mathbf{u} \in \mathrm{co}\mathcal{X} \cap \mathrm{co}\mathcal{Y} : \mathbf{u} = \sum_i \alpha_{x,i}\mathbf{x}_i = \sum_j \alpha_{y,j}\mathbf{y}_j \,. \tag{1}$$

A linear separator v, described by the parameters $\mathbf{w}$ and w_0, must satisfy $\forall \mathbf{x} \in \mathrm{co}\mathcal{X} : v(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0 \geq 0$ and $\forall \mathbf{y} \in \mathrm{co}\mathcal{Y} : v(\mathbf{x}) = \mathbf{w}^\top \mathbf{y} + w_0 < 0$. The linear model and the convex hull can directly be combined and for a point $\mathbf{x}$ in the convex hull $\mathrm{co}\mathcal{X}$ we get

$$v(\mathbf{x}) = \sum_i \mathbf{w}^\top (\alpha_{x,i}\mathbf{x}_i) + w_0 = \sum_i \alpha_{x,i} \left(\mathbf{w}^\top \mathbf{x}_i + w_0 \right) \,. \tag{2}$$

Consider $y(\mathbf{u})$:

$$y(\mathbf{u}) = \sum_i \alpha_{x,i} \left(\mathbf{w}^\top \mathbf{x}_i + w_0 \right) = \sum_j \alpha_{y,i} \left(\mathbf{w}^\top \mathbf{y}_j + w_0 \right) \,, \tag{3}$$

which cannot be true if the sets $\mathcal{X}$ and $\mathcal{Y}$ are linearly separable. ■

1.2 Problem 2

Given is a linearly separable dataset with two classes C_1 and C_2. The weight vector with the bias term included separates the classes, where the value at the points of the decision boundary $\mathbf{x}_*$ is $\mathbf{w}^\top \mathbf{x}_* = 0$. Setting the magnitude to infinity yields the following function

$$\lim_{|\mathbf{w}| \to \infty} \mathbf{w}^\top \mathbf{x} = \begin{cases} \infty, & \text{if } \mathbf{x} \in C_1 \\ -\infty, & \text{if } \mathbf{x} \in C_2 \\ 0, & \mathbf{x} \text{ is on the boundary.} \end{cases} \tag{4}$$

Taking the sigmoid function of $\mathbf{w}^\top \mathbf{x}$ gives a unit step function. In order to prevent this degeneration, regularization on the elements of $\mathbf{w}$ is required.

1.3 Problem 3

$$\phi(x_1, x_2) = \arctan \frac{x_2}{x_1} \,. \tag{5}$$

2 Basis Functions

2.1 Problem 4

For every point $\mathbf{x}$ on the decision boundary we have

$$w_0 + w_1 x_1 + x_2 x_2 = 0 \,. \tag{6}$$

Using the given intersection points with the principal axes, we get

$$w_0 + 2w_1 = 0 \tag{7}$$

and

$$w_0 + 5w_2 = 0 \,. \tag{8}$$

Choosing $w_0 = -10$ yields $w_1 = 5$ and $w_2 = 2$. Finally, we get the parameter vector

$$\mathbf{w} = \begin{pmatrix} -10 \\ 5 \\ 2 \end{pmatrix} \,. \tag{9}$$

Machine Learning Worksheet 06

Optimization

1 Convexity

Problem 1: Prove or disprove whether the following functions are convex on the given set D:

i) $f(x, y, z) = 3x + e^{y+z} - min\{-x^2, log(y)\}$ and $D = (-100, 100) \times (1, 50) \times (10, 20)$

ii) $f(x, y) = y \cdot x^3 - 2 \cdot y \cdot x^2 + y + 4$ and $D = (-10, 10) \times (-10, 10)$

iii) $f(x) = log(x) + x^3$ and $D = (1, \infty)$

iv) $f(x) = -\min(2\log(2x), -x^2 + 4x - 32)$ and $D = \mathbb{R}^+$

Problem 2: Prove the following statement: Let $f_1 : \mathbb{R}^d \to \mathbb{R}$ and $f_2 : \mathbb{R}^d \to \mathbb{R}$ be convex functions, then $h(x) := f_1(x) + f_2(x)$ is a also convex function.

Problem 3: Given two convex functions $f_1 : \mathbb{R} \to \mathbb{R}$ and $f_2 : \mathbb{R} \to \mathbb{R}$, prove or disprove that the function $g(x) = f_1(x) \cdot f_2(x)$ is also convex.

2 Minimization of convex functions

Problem 4: Prove that for convex functions each local minimum is a global minimum. More specifically, given a convex function $f : \mathbb{R}^N \to \mathbb{R}$, prove that if $\nabla f(\theta^*) = 0$ then θ^* is a global minimum.

3 Gradient Descent

Problem 5: Load the notebook `06_hw_optimization_logistic_regression.ipynb` from Piazza. Fill in the missing code and run the notebook. Convert the evaluated notebook to pdf and add it to the printout of your homework.

Homework 06 - Optimization

IN2064 Machine Learning

Arne Sachtler - *Registration Number: 03692662*

December 10, 2017

1 Convexity

1.1 Problem 1

- **i)**
  Given the function

$$f(x, y, z) = 3x + e^{y+z} - \min\{-x^2, \log(y)\} \text{ on } D = (-100, 100) \times (1, 50) \times (10, 20) \tag{1}$$

  f can be decomposed into a sum of three functions:

$$f(x, y, z) = g(x) + h(y, z) + k(x, y)\,, \tag{2}$$

  where

  1. $g(x) = 3x$
  2. $h(y, z) = e^{y+z}$
  3. $k(x, y) = -\min(-x^2, \log(y))$.

  The sum of functions is a convex function if the summands are convex. Here, g is a linear function and linear functions are both convex and concave. The exponential map h is convex, too. The interesting part is k, if it can be proven that k is convex it can be concluded that f is convex. Using the equation $-\min(x, y) = \max(-x, -y)$ the function k can be rewritten to

$$k(x, y) = -\min(-x^2, \log(y)) = \max(x^2, -\log(y))\,. \tag{3}$$

  The maximum of convex functions is also convex. We can follow that k is convex as x^2 is convex on $(-100, 100)$ (and everywhere else) and $-log(y)$ is convex on $(1, 50)$. Therefore, f is convex. ■

- **ii)**
  Given the function

$$f(x, y) = yx^3 - 2yx^2 + y + 4 \text{ on } D = (-10, 10)^2 \tag{4}$$

we can compute the Hessian matrix in order to disprove convexity. First the gradient:

$$\nabla f(x,y) = \begin{pmatrix} 3x^2y - 2xy & x^3 - 2x^2 + 1 \end{pmatrix}, \tag{5}$$

then the Hessian is

$$\mathbf{H} = \begin{pmatrix} 6xy - 2y & 3x^2 - 2x \\ 3x^2 - 2x & 0 \end{pmatrix}. \tag{6}$$

A function f is only convex if its Hessian is positive semi-definite. A matrix is positive semi-definite iff the eigenvalues are greater than or equal to zero. In order to disprove convexity we compute the eigenvalues of the Hessian using the characteristic polynomial

$$\det(\mathbf{H} - \lambda\mathbf{I}) = ... = \lambda^2 + (2y - 6xy)\lambda - (9x^4 - 12x^3 + 4x^2). \tag{7}$$

For the eigenvalues we get

$$\lambda_{1,2} = -\frac{2y - 6xy}{2} \pm \sqrt{\frac{(2y - 6xy)^2}{4} + 9x^4 - 12x^3 + 4x^2}. \tag{8}$$

Consider the point $(0,1) \in D$. For the eigenvalues of the Hessian we get $\lambda_1 = 0$ and $\lambda_2 = -2 < 0$. It is proven that f is not convex as $\mathbf{H}$ is not positive semi-definite. ■

- **iii)**
  Consider

$$f(x) = \log(x) + x^3 \text{ on } D = (1, \infty). \tag{9}$$

To prove convexity, the second derivative can be computed:

$$f'(x) = \frac{1}{x} + 3x^2 \tag{10}$$

and

$$f''(x) = 6x - \frac{1}{x^2}. \tag{11}$$

As it holds that

$$\forall x \in (1, \infty) : 6x - \frac{1}{x^2} > 0 \tag{12}$$

we conclude that f is convex. ■

- **iv)** Similar to the first problem, the given function

$$f(x) = -\min(2\log(2x), -x^2 + 4x - 32) \text{ on } D = \mathbb{R}^+ \tag{13}$$

can be rewritten in terms of the maximum function

$$f(x) = \max(-2\log(2x), x^2 - 4x + 32). \tag{14}$$

Again, the maximum of two functions is convex if all functions that are considered for the maximum are convex. The function $g(x) = x^2 - 4x + 32$ is convex as $g''(x) = 2 > 0$. For $h(x) = -2\log(2x)$ we get $h'(x) = -\frac{2}{x}$ and $h''(x) = \frac{2}{x^2} \geq 0 \forall x \in \mathbb{R}^+$. Therefore, f is convex. ■

1.2 Problem 2

Let $f_1 : \mathbb{R}^d \to \mathbb{R}$ and $f_2 : \mathbb{R}^d \to \mathbb{R}$ be convex. Therefore it holds that (by the definition of convexity):

$$\forall f \in \{f_1, f_2\} \, \forall x, y \in \mathbb{R}^d : \lambda f(x) + (1-\lambda) f(y) \geq f(\lambda x + (1-\lambda) y) \text{ for } \lambda \in [0,1]\,. \quad (15)$$

The objective is to prove that $h(x) = f_1(x) + f_2(x)$ is convex. This can be proven straightforwardly by using the definition of convexity

$$\begin{aligned} \lambda h(x) + (1-\lambda) h(y) &= \lambda(f_1(x) + f_2(x)) + (1-\lambda)(f_1(y) + f_2(y)) && (16)\\ &= \lambda f_1(x) + (1-\lambda) f_1(y) + \lambda f_2(x) + (1-\lambda) f_2(y) && (17)\\ &\geq f_1(\lambda x + (1-\lambda) y) + f_2(\lambda x + (1-\lambda) y) && (18)\\ &= h(\lambda x + (1-\lambda) y). && (19) \end{aligned}$$

■

1.3 Problem 3

Consider the following convex functions on $\mathbb{R}$

- $f_1(x) = x^2$
- $f_2(x) = x$.

$f_1(x)$ is convex as $f_1''(x) = 2 > 0$ and f_2 is convex as we know that all linear functions are both convex and concave. Now consider the function $g(x)$, which is the product of f_1 and f_2. We obtain

$$g(x) = f_1(x) f_2(x) = x^2 \cdot x = x^3\,. \quad (20)$$

The second derivative $g''(x) = 6x$ is negative for $x \in \mathbb{R}^-$. Therefore, we found one example for a non-convex function, which is the product of convex functions and we conclude that the product of convex functions is in general not necessarily convex. ■

2 Minimization of Convex Functions

2.1 Problem 4

We can argue by contradiction. Consider the global minimum θ_g and a (hypothetical) local minimum θ_l where $\theta_g \neq \theta_l$ and (wlog) $\theta_l < \theta_g$. As θ_g and θ_l are both minima, the second derivative is positive those points. Hence, the functions leaves both minima with positive slope in positive θ-direction. Therefore, it must hold that $\exists \tilde{\theta} \in (\theta_l, \theta_g) : f(\tilde{\theta}) > f(\theta_l) \wedge f(\tilde{\theta}) > f(\theta_g)$. There must be a point $\tilde{\theta}$ in between the local and the global minimum where the function value is greater than those a the minima. This is a contradiction as f is convex. ■

3 Gradient Descent

See the pdf pages attached.

Programming assignment 6: Optimization: Logistic regression

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
```

Your task

In this notebook code skeleton for performing logistic regression with gradient descent is given. Your task is to complete the functions where required. You are only allowed to use built-in Python functions, as well as any `numpy` functions. No other libraries / imports are allowed.

For numerical reasons, we actually minimize the following loss function

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} NLL(\mathbf{w}) + \lambda ||\mathbf{w}||_2^2$$

where $NLL(\mathbf{w})$ is the negative log-likelihood function, as defined in the lecture (Eq. 33)

Load and preprocess the data

In this assignment we will work with the UCI ML Breast Cancer Wisconsin (Diagnostic) dataset https://goo.gl/U2Uwz2 (https://goo.gl/U2Uwz2).

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. There are 212 malignant examples and 357 benign examples.

In [30]:
```
X, y = load_breast_cancer(return_X_y=True)

# Add a vector of ones to the data matrix to absorb the bias term
X = np.hstack([np.ones([X.shape[0], 1]), X])

# Set the random seed so that we have reproducible experiments
np.random.seed(123)


# Split into train and test
test_size = 0.3
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
```

Task 1: Implement the sigmoid function

In [3]:
```
def sigmoid(t):
    """
    Applies the sigmoid function elementwise to the input data.

    Parameters
    ----------
    t : array, arbitrary shape
        Input data.

    Returns
    -------
    t_sigmoid : array, arbitrary shape.
        Data after applying the sigmoid function.
    """
    return 1/(1+np.exp(-t))
```

Task 2: Implement the negative log likelihood

As defined in Eq. 33

In [4]:
```
def negative_log_likelihood(X, y, w):
    """
    Negative Log Likelihood of the Logistic Regression.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Classification targets.
    w : array, shape [D]
        Regression coefficients (w[0] is the bias term).

    Returns
    -------
    nll : float
        The negative log likelihood.
    """
    return -np.sum(y*np.log(sigmoid(X.dot(w))) + (np.ones_like(y)-y)*np.log(np.
ones_like(y) - sigmoid(X.dot(w))))
```

Computing the loss function $\mathcal{L}(\mathbf{w})$ (nothing to do here)

In [5]:
```
def compute_loss(X, y, w, lmbda):
    """
    Negative Log Likelihood of the Logistic Regression.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Classification targets.
    w : array, shape [D]
        Regression coefficients (w[0] is the bias term).
    lmbda : float
        L2 regularization strength.

    Returns
    -------
    loss : float
        Loss of the regularized logistic regression model.
    """
    # The bias term w[0] is not regularized by convention
    return negative_log_likelihood(X, y, w) / len(y) + lmbda * np.linalg.norm(w
[1:])
```

Task 3: Implement the gradient $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})$

Make sure that you compute the gradient of the loss function $\mathcal{L}(\mathbf{w})$ (not simply the NLL!)

In [39]:
```
def get_gradient(X, y, w, mini_batch_indices, lmbda):
    """
    Calculates the gradient (full or mini-batch) of the negative log likelihood
w.r.t. w.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Classification targets.
    w : array, shape [D]
        Regression coefficients (w[0] is the bias term).
    mini_batch_indices: array, shape [mini_batch_size]
        The indices of the data points to be included in the (stochastic) calcu
lation of the gradient.
        This includes the full batch gradient as well, if mini_batch_indices =
np.arange(n_train).
    lmbda: float
        Regularization strentgh. lmbda = 0 means having no regularization.

    Returns
    -------
    dw : array, shape [D]
        Gradient w.r.t. w.
    """
    return -(y[mini_batch_indices] - sigmoid(X[mini_batch_indices,:].dot(w))).d
ot(X[mini_batch_indices,:])/len(mini_batch_indices) + 2*lmbda*w
```

Train the logistic regression model (nothing to do here)

In [7]:
```
def logistic_regression(X, y, num_steps, learning_rate, mini_batch_size, lmbda,
verbose):
    """
    Performs logistic regression with (stochastic) gradient descent.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Classification targets.
    num_steps : int
        Number of steps of gradient descent to perform.
    learning_rate: float
        The learning rate to use when updating the parameters w.
    mini_batch_size: int
        The number of examples in each mini-batch.
        If mini_batch_size=n_train we perform full batch gradient descent.
    lmbda: float
        Regularization strentgh. lmbda = 0 means having no regularization.
    verbose : bool
        Whether to print the loss during optimization.

    Returns
    -------
    w : array, shape [D]
        Optimal regression coefficients (w[0] is the bias term).
    trace: list
        Trace of the loss function after each step of gradient descent.
    """

    trace = [] # saves the value of loss every 50 iterations to be able to plot
it later
    n_train = X.shape[0] # number of training instances

    w = np.zeros(X.shape[1]) # initialize the parameters to zeros

    # run gradient descent for a given number of steps
    for step in range(num_steps):
        permuted_idx = np.random.permutation(n_train) # shuffle the data

        # go over each mini-batch and update the paramters
        # if mini_batch_size = n_train we perform full batch GD and this loop r
uns only once
        for idx in range(0, n_train, mini_batch_size):
            # get the random indices to be included in the mini batch
            mini_batch_indices = permuted_idx[idx:idx+mini_batch_size]
            gradient = get_gradient(X, y, w, mini_batch_indices, lmbda)
            # update the parameters
            w = w - learning_rate * gradient
        # calculate and save the current loss value every 50 iterations
        if step % 50 == 0:
            loss = compute_loss(X, y, w, lmbda)
            trace.append(loss)
            # print loss to monitor the progress
            if verbose and step % 50 == 0:
                print('Step {0}, loss = {1:.4f}'.format(step, loss))

    return w, trace
```

Task 4: Implement the function to obtain the predictions

In [8]:
```
def predict(X, w):
    """
    Parameters
    ----------
    X : array, shape [N_test, D]
        (Augmented) feature matrix.
    w : array, shape [D]
        Regression coefficients (w[0] is the bias term).

    Returns
    -------
    y_pred : array, shape [N_test]
        A binary array of predictions.
    """
    return sigmoid(X.dot(w)) >= 0.5
```

Full batch gradient descent

In [9]:
```
# Change this to True if you want to see loss values over iterations.
verbose = False
```

In [36]:
```
n_train = X_train.shape[0]
w_full, trace_full = logistic_regression(X_train,
                                         y_train,
                                         num_steps=8000,
                                         learning_rate=1e-5,
                                         mini_batch_size=n_train,
                                         lmbda=0.1,
                                         verbose=verbose)
```

In [37]:
```
n_train = X_train.shape[0]
w_minibatch, trace_minibatch = logistic_regression(X_train,
                                                   y_train,
                                                   num_steps=8000,
                                                   learning_rate=1e-5,
                                                   mini_batch_size=50,
                                                   lmbda=0.1,
                                                   verbose=verbose)
```

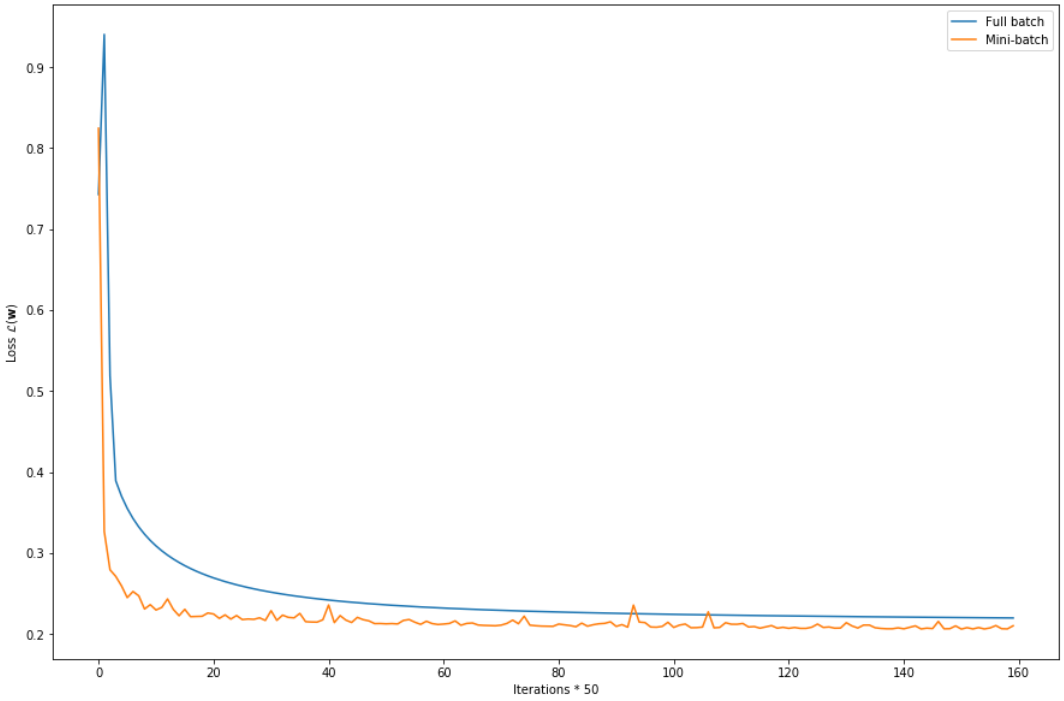
Our reference solution produces, but don't worry if yours is not exactly the same.

```
Full batch: accuracy: 0.9240, f1_score: 0.9384
Mini-batch: accuracy: 0.9415, f1_score: 0.9533
```

In [38]:
```
y_pred_full = predict(X_test, w_full)
y_pred_minibatch = predict(X_test, w_minibatch)

print('Full batch: accuracy: {:.4f}, f1_score: {:.4f}'
      .format(accuracy_score(y_test, y_pred_full), f1_score(y_test, y_pred_full
)))
print('Mini-batch: accuracy: {:.4f}, f1_score: {:.4f}'
      .format(accuracy_score(y_test, y_pred_minibatch), f1_score(y_test, y_pred
_minibatch)))
```

```
Full batch: accuracy: 0.9240, f1_score: 0.9384
Mini-batch: accuracy: 0.9415, f1_score: 0.9528
```

In [40]:
```
plt.figure(figsize=[15, 10])
plt.plot(trace_full, label='Full batch')
plt.plot(trace_minibatch, label='Mini-batch')
plt.xlabel('Iterations * 50')
plt.ylabel('Loss $\mathcal{L}(\mathbf{w})$')
plt.legend()
plt.show()
```



Machine Learning Worksheet 07

Constrained Optimization and SVM

1 Constrained Optimization

Problem 1: Solve the following constrained optimization problem using the recipe described in the lecture (slide 31).

$$\begin{aligned} \text{minimize} \quad & f_0(\boldsymbol{x}) = -(x_1 + x_2) \\ \text{subject to} \quad & f_1(\boldsymbol{x}) = x_1^2 + x_2^2 - 1 \leq 0 \end{aligned}$$

2 SVM

Problem 2: Explain the similarities and differences between the SVM and perceptron algorithms.

Problem 3: Show that the duality gap is zero for SVM.

Problem 4: Recall, that the dual function for SVM (slide 37) can be written as

$$g(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{Q} \boldsymbol{\alpha} + \boldsymbol{\alpha}^T \mathbf{1}_N$$

(a) Show how the matrix $\boldsymbol{Q}$ can be computed. *(Hint: You might want to use Hadamard product, denoted as $\odot$).*

(b) Prove that the matrix $\boldsymbol{Q}$ is negative (semi-)definite.

(c) Explain, what the negative (semi-)definiteness means for our optimization problem. Why is this property important?

Problem 5: Load the notebook `07_homework_svm.ipynb` from Piazza. Fill in the missing code and run the notebook. Convert the evaluated notebook to pdf and add it to the printout of your homework.

Note: We suggest that you use Anaconda for installing Python and Jupyter, as well as for managing packages. We recommend that you use Python 3.

For more information on Jupyter notebooks and how to convert them to other formats, consult the Jupyter documentation and nbconvert documentation.

Homework 07 - Constrained Optimization and SVM

IN2064 Machine Learning

Arne Sachtler - *Registration Number: 03692662*

December 11, 2017

1 Constrained Optimization

1.1 Problem 1

Given constrained optimization problem:

$$\begin{aligned} \text{minimize} \quad f_0(\mathbf{x}) &= -(x_1 + x_2) && (1) \\ \text{subject to} \quad f_1(\mathbf{x}) &= x_1^2 + x_2^2 - 1 \leq 0\,. && (2) \end{aligned}$$

Step 1: Calculate the Lagrangian

$$L(\boldsymbol{x}, \alpha) = -(x_1 + x_2) + \alpha(x_1^2 + x_2^2 - 1) \tag{3}$$

Step 2: Obtain the Lagrange dual function
First, in order to get the minimum, we compute the derivative with respect to $\mathbf{x}$

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \alpha) = \begin{pmatrix} 2\alpha x_1 - 1 & 2\alpha x_2 - 1 \end{pmatrix}\,. \tag{4}$$

Setting it to zero yield $\mathbf{x}^*$ being

$$\mathbf{x}^* = \begin{pmatrix} \frac{1}{2\alpha} & \frac{1}{2\alpha} \end{pmatrix}^\top\,. \tag{5}$$

And for the Lagrange dual function we get

$$g(\alpha) = -\frac{1}{2\alpha} - \alpha\,. \tag{6}$$

Step 3: Solve the dual problem
In order to solve the dual problem we compute the derivative of the Lagrange dual function with respect to α

$$\frac{dg}{d\alpha} = \frac{1}{2\alpha^2} - 1\,. \tag{7}$$

Setting it to zeros yields $\alpha = \pm\frac{1}{4}$ and we only take $\alpha = \frac{1}{4}$ as we constrain $\alpha \geq 0$ for the dual problem. Then, the obtained α can be inserted into (5) and we conclude the final optimum at

$$\mathbf{x}_{optimal} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} . \tag{8}$$

2 SVM

2.1 Problem 2

The perceptron and the SVM both use a linear hyperplane to separate classes. Thus, they are (without further techniques) only applicable for linear separable data. While the perceptron usually assigns class labels of 0 and 1 a support vector machine usually uses 1 and -1 as labels. The SVM chooses the class separating hyperplane such that the margin (the distance to the separating plane where no training data point is located) is maximized. In contrast, the perceptron just uses any hyperplane separating the data. The perceptron uses a simple iterative learning rule, where for every misclassified sample the weights and biases are updated until all data points are correctly classified. On the other hand, SVM required quadratic programming optimization.

2.2 Problem 3

The duality gap is zero as $f_0(\mathbf{w}, b) = \frac{1}{2}\mathbf{w}^\top\mathbf{w}$ is convex and the constraints are affine. ■

2.3 Problem 4

We are given the general formula for the dual function

$$g(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{Q}\boldsymbol{\alpha} + \boldsymbol{\alpha}^\top \mathbf{1}_N . \tag{9}$$

The dual function for support vectors machines is also given:

$$g(\boldsymbol{\alpha}) = \sum_{i=1} N\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_1 y_2 \alpha_1 \alpha_2 \mathbf{x}_i^\top \mathbf{x}_j . \tag{10}$$

(a)
For $\mathbf{Q}$ we see that is must be

$$\mathbf{Q} = -\begin{pmatrix} y_1 y_1 \mathbf{x}_1^\top \mathbf{x}_1 & y_1 y_2 \mathbf{x}_1^\top \mathbf{x}_2 & \dots & y_1 y_N \mathbf{x}_1^\top \mathbf{x}_N \\ y_2 y_1 \mathbf{x}_2^\top \mathbf{x}_1 & y_2 y_2 \mathbf{x}_2^\top \mathbf{x}_2 & \dots & y_2 y_N \mathbf{x}_2^\top \mathbf{x}_N \\ \vdots & \vdots & \ddots & \vdots \\ y_N y_1 \mathbf{x}_1^\top \mathbf{x}_1 & y_N y_2 \mathbf{x}_N^\top \mathbf{x}_2 & \dots & y_N y_N \mathbf{x}_N^\top \mathbf{x}_N \end{pmatrix} \tag{11}$$

We observe an element-wise product of $\mathbf{X}^\top\mathbf{X}$ and the outer product of $\mathbf{y}$. Thus, we can rewrite $\mathbf{Q}$ to:

$$\mathbf{Q} = -\mathbf{y}\mathbf{y}^\top \odot \mathbf{X}^\top\mathbf{X} . \tag{12}$$

(b)
In order to prove that $\mathbf{Q}$ is negative semi-definite we can prove that $\mathbf{P} = -\mathbf{Q}$ is positive semi-definite.

$$\mathbf{P} = -\mathbf{Q} = \mathbf{y}\mathbf{y}^\top \odot \mathbf{X}^\top \mathbf{X}\,. \tag{13}$$

The matrix computed by the element-wise product is positive semi-definite if the factors are, which is the case for these quadratic forms. Therefore $\mathbf{Q}$ is negative semi-definite. ■

(c)
The negative semi-definiteness of $\mathbf{Q}$ means that the quadratic form $\frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{Q}\boldsymbol{\alpha}$ is a *concave* function. For concave functions every local maximum is a global maximum. Thus our solution for $\boldsymbol{\alpha}$ will be a global maximum under concavity.

2.4 Problem 5

See the pages attached.

07_homework_svm

December 10, 2017

1 Programming assignment 7: SVM

```
In [89]: import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline

         from sklearn.datasets import make_blobs

         from cvxopt import matrix, solvers
```

1.1 Your task

In this sheet we will implement a simple binary SVM classifier.

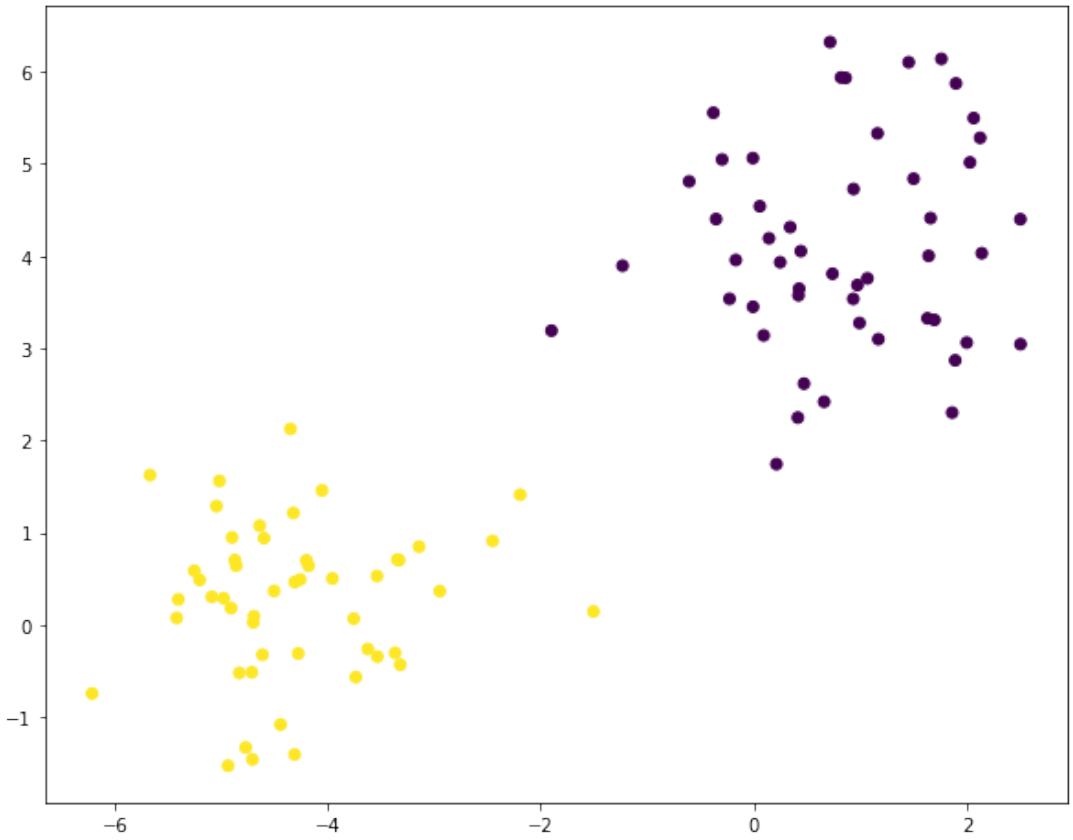
We will use `CVXOPT` http://cvxopt.org/ - a Python library for convex optimization. If you use `Anaconda`, you can install it using

```
conda install cvxopt
```

As usual, your task is to fill out the missing code, run the notebook, convert it to PDF and attach it you your HW solution.

1.2 Generate and visualize the data

```
In [90]: N = 100  # number of samples
         D = 2  # number of dimensions
         C = 2  # number of classes
         seed = 3  # for reproducible experiments

         X, y = make_blobs(n_samples=N, n_features=D, centers=2, random_state=seed)
         y[y == 0] = -1  # it is more convenient to have {-1, 1} as class labels (instead of {
         y = y.astype(np.float)
         plt.figure(figsize=[10, 8])
         plt.scatter(X[:, 0], X[:, 1], c=y)
         plt.show()
```



1.3 Task 1: Solving the SVM dual problem

Remember, that the SVM dual problem can be formulated as a Quadratic programming (QP) problem. We will solve it using a QP solver from the `CVXOPT` library.

The general form of a QP is

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x}$$

$$\text{subject to} \quad \mathbf{G}\mathbf{x} \preceq \mathbf{h}$$

$$\text{and} \quad \mathbf{A}\mathbf{x} = \mathbf{b}$$

where $\preceq$ denotes "elementwise less than or equal to".

Your task is to formulate the SVM dual problems as a QP and solve it using `CVXOPT`, i.e. specify the matrices $\mathbf{P}, \mathbf{G}, \mathbf{A}$ and vectors $\mathbf{q}, \mathbf{h}, \mathbf{b}$.

```
In [138]: def solve_dual_svm(X, y):
              """Solve the dual formulation of the SVM problem.

              Parameters
```

```
    ----------
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).

    Returns
    -------
    alphas : array, shape [N]
        Solution of the dual problem.
    """
    # TODO
    # These variables have to be of type cvxopt.matrix
    n = X.shape[0]

    P = matrix(np.outer(y,y) * X.dot(X.T))
    q = matrix(-1.0, (n,1))
    G = matrix(-np.eye(n))
    h = matrix(0.0, (n,1))
    A = matrix(y[:,np.newaxis].T)
    b = matrix(0.0)
    solvers.options['show_progress'] = False
    solution = solvers.qp(P, q, G, h, A, b)
    alphas = np.array(solution['x'])
    return alphas
```

1.4 Task 2: Recovering the weights and the bias

```
In [141]: def compute_weights_and_bias(alpha, X, y):
    """Recover the weights w and the bias b using the dual solution alpha.

    Parameters
    ----------
    alpha : array, shape [N]
        Solution of the dual problem.
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).

    Returns
    -------
    w : array, shape [D]
        Weight vector.
    b : float
        Bias term.
    """
    w = X.T.dot((alpha*y[:,np.newaxis]))
```

```
    b = y[np.argmax(alpha)]-w.T.dot(X[np.argmax(alpha),:])
    return w, b
```

1.5 Visualize the result (nothing to do here)

```
In [99]: def plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b):
    """Plot the data as a scatter plot together with the separating hyperplane.

    Parameters
    ----------
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).
    alpha : array, shape [N]
        Solution of the dual problem.
    w : array, shape [D]
        Weight vector.
    b : float
        Bias term.
    """
    plt.figure(figsize=[10, 8])
    # Plot the hyperplane
    slope = -w[0] / w[1]
    intercept = -b / w[1]
    x = np.linspace(X[:, 0].min(), X[:, 0].max())
    plt.plot(x, x * slope + intercept, 'k-', label='decision boundary')
    # Plot all the datapoints
    plt.scatter(X[:, 0], X[:, 1], c=y)
    # Mark the support vectors
    support_vecs = (alpha > 1e-4).reshape(-1)
    plt.scatter(X[support_vecs, 0], X[support_vecs, 1], c=y[support_vecs],
                s=250, marker='*', label='support vectors')
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.legend(loc='upper left')
```

The reference solution is

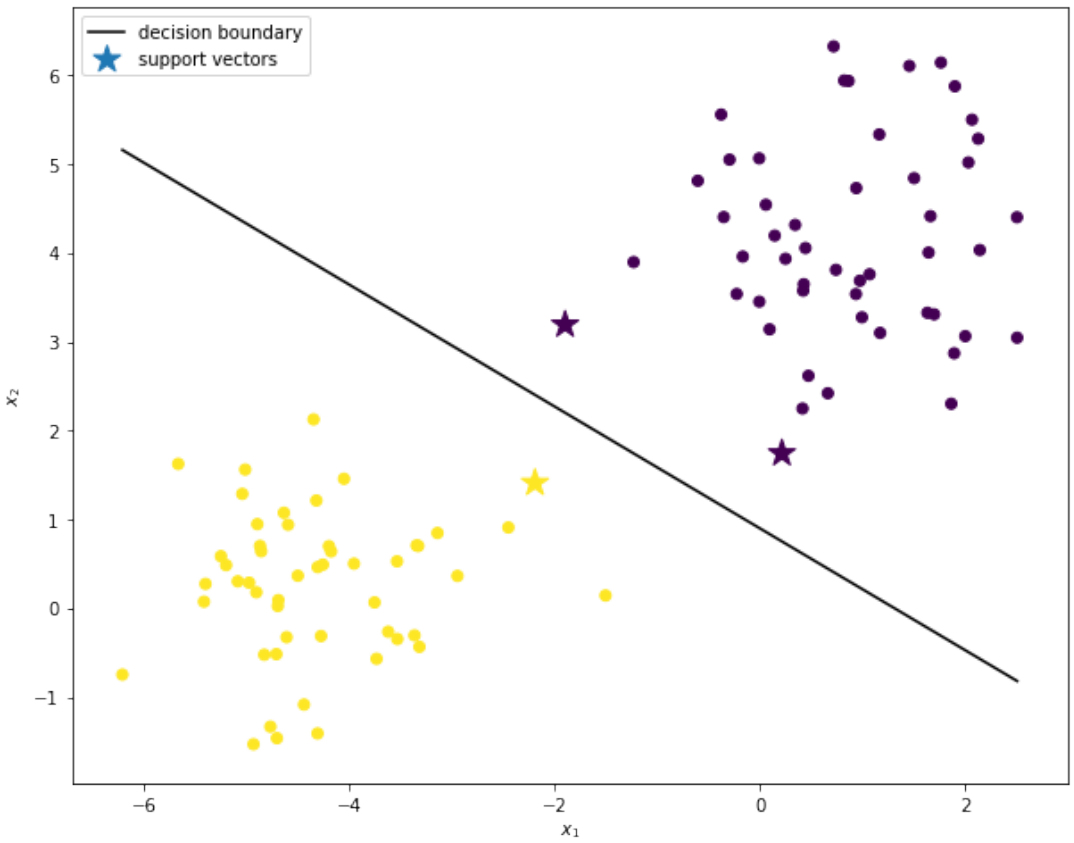
```
w = array([[-0.69192638],
           [-1.00973312]])

b = 0.907667782
```

Indices of the support vectors are

```
[38, 47, 92]
```

```
In [142]: alpha = solve_dual_svm(X, y)
          w, b = compute_weights_and_bias(alpha, X, y)
          print("w: {}".format(w))
          print("b: {}".format(b))
          plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b)
          plt.show()
```

```
w: [[-0.69192638]
 [-1.00973312]]
b: [ 0.90766774]
```



Machine Learning Worksheet 08

Soft-margin SVM and Kernels

1 Soft-margin SVM

Problem 1: Assume that we have a linearly separable dataset $\mathcal{D}$, on which a soft-margin SVM is fitted. Is it guaranteed that all training samples in $\mathcal{D}$ will be assigned the correct label by the fitted model? Explain your answer.

Problem 2: Why do we need to ensure that $C > 0$ in the slack variable formulation of soft-margin SVM? What would happen is this was not the case?

2 Kernels

Problem 3: Show that for $c \geq 0$ and $d \in \mathbb{N}^+$ the function

$$K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y} + c)^d$$

is a valid kernel.

3 Gaussian kernel

Problem 4: Let us define a feature transformation $\boldsymbol{\phi_n} : \mathbb{R} \to \mathbb{R}^n$ as follows:

$$\boldsymbol{\phi_n}(x) = \left\{ e^{-x^2/2\sigma^2}, e^{-x^2/2\sigma^2}\frac{x}{\sigma}, \frac{e^{-x^2/2\sigma^2}\left(\frac{x}{\sigma}\right)^2}{\sqrt{2}}, \ldots, \frac{e^{-x^2/2\sigma^2}\left(\frac{x}{\sigma}\right)^i}{\sqrt{i!}}, \ldots, \frac{e^{-x^2/2\sigma^2}\left(\frac{x}{\sigma}\right)^n}{\sqrt{n!}} \right\}$$

Suppose we let $n \to \infty$ and define a new feature transformation:

$$\boldsymbol{\phi_\infty}(x) = \left\{ e^{-x^2/2\sigma^2}, e^{-x^2/2\sigma^2}\frac{x}{\sigma}, \frac{e^{-x^2/2\sigma^2}\left(\frac{x}{\sigma}\right)^2}{\sqrt{2}}, \ldots, \frac{e^{-x^2/2\sigma^2}\left(\frac{x}{\sigma}\right)^i}{\sqrt{i!}}, \ldots \right\}$$

Can we directly apply this feature transformation to data? Explain why or why not!

Problem 5: From the lecture, we know that we can express a linear classifier using only inner products of input vectors in the transformed feature space. It would be great if we could somehow use the feature space obtained by the feature transformation $\boldsymbol{\phi_\infty}$. However, to do this, we must be able to compute the

inner product of samples in this infinite feature space. We define the inner product between two *infinite* vectors $\boldsymbol{\phi_\infty}(x)$ and $\boldsymbol{\phi_\infty}(y)$ as the infinite sum given in the following equation:

$$K(x,y) = \sum_{i=0}^{\infty} \boldsymbol{\phi_{\infty,i}}(x)\, \boldsymbol{\phi_{\infty,i}}(y)$$

What is the explicit form of $K(x,y)$? (Hint: Think of the Taylor series of e^x.) With such a high dimensional feature space, should we be concerned about overfitting?

Problem 6: Can any *finite* set of points be linearly separated in the feature space defined by $\boldsymbol{\phi_\infty}$ if σ can be chosen freely?

4 Kernelized k-nearest neighbors

To classify the point $\boldsymbol{x}$ the k-nearest neighbors finds the k training samples $\mathcal{N} = \{\boldsymbol{x}^{(s_1)}, \boldsymbol{x}^{(s_2)}, \ldots, \boldsymbol{x}^{(s_k)}\}$ that have the shortest distance $||\boldsymbol{x} - \boldsymbol{x}^{(s_i)}||_2$ to $\boldsymbol{x}$. Then the label that is mostly represented in the neighbor set $\mathcal{N}$ is assigned to $\boldsymbol{x}$.

Problem 7: Formulate the k-nearest neighbors algorithm in feature space by introducing the feature map $\boldsymbol{\phi}(\boldsymbol{x})$. Then rewrite the k-nearest neighbors algorithm so that it only depends on the scalar product in feature space $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{y})$.

Homework 08 - Soft-margin SVM and Kernels

IN2064 Machine Learning

Arne Sachtler - *Registration Number: 03692662*

December 18, 2017

1 Soft-margin SVM

1.1 Problem 1

Using a soft-margin support vector machine ensures, that the α's cannot go to infinity as the are constraint to lie within 0 and C. In the hard-margin case the support vector machine can overfit as soon as a single outlier appears. The soft-margin SVM does not guarantee that all training sample are assigned with the correct label as outlier may lie on the wrong side of the soft-margin SVM plane.

1.2 Problem 2

The first observation is, that a choice of $C < 0$ is definitely a bad one. The objective function for soft-margin SVM's is

$$f_0(\mathbf{w}, b, \xi) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C \sum \xi_i \,, \tag{1}$$

which is minimized. Therefore the sum over the ξ_i's is maximized and the SVM tries to assign the labels as bad as possible. It would flip the decision boundary such that the normal vector of the decision hyperplane is inverted.

2 Kernels

2.1 Problem 3

Task: Show that for $c \geq 0$ and $d \in \mathbb{N}^+$

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y} + c)^d \tag{2}$$

is a valid kernel.

Using Binomial law the polynomial in the function K can be rewritten to

$$\begin{aligned} K(\mathbf{x},\mathbf{y}) &= (\mathbf{x}^\top \mathbf{y} + c)^d && (3)\\ &= \sum_{k=0}^{d} \binom{d}{k} \left(\sum_i x_i y_i \right)^{d-k} c^k && (4)\\ &= \sum_{k=0}^{d} \sum_i \binom{d}{k} c^k x_i^{d-k} y_i^{d-k} . && (5) \end{aligned}$$

Observing the latter expression already looks like a scalar product. One can define a feature map φ where the function K is the dot product. This feature map can be designed straightforwardly from the developed expression.

$$\varphi(\mathbf{x}) = \begin{pmatrix} x_1^d & \dots & x_n^d & \sqrt{dc}\, x_1^{d-1} & \dots & \sqrt{dc}\, x_n^{d-1} & \sqrt{\binom{d}{2} c^2}\, x_1^{d-2} & \dots & \sqrt{c^d}\, x_n \end{pmatrix}^\top \quad (6)$$

Consequently,

$$\varphi(\mathbf{x})^\top \varphi(\mathbf{y}) = K(\mathbf{x},\mathbf{y}) = (\mathbf{x}^\top \mathbf{y} + c)^d \quad (7)$$

and K is a valid kernel as it is a scalar product in the feature space $\varphi(\mathbf{x})$. ■

3 Gaussian Kernel

3.1 Problem 4

No, we can't. The feature map $\varphi_\infty(x)$ transforms into an infinite-dimensional feature space. That can never be computed in finite time and stored in a finite amount of memory. The computation of the feature map $\varphi_\infty(x)$ is simply physically impossible.

3.2 Problem 5

We simply try to express the dot-product of the infinite-dimensional feature space using the sum-expression. Afterwards, simplifications and a Taylor series yield a closed-form result for the dot-product of two infinite-dimensional vectors. The Taylor series of the exponential function

$$\exp(x) = \sum_{i=0}^{\infty} \frac{x^i}{i!} \quad (8)$$

is used.

We start with the expression for the dot-product

$$\begin{aligned}
\varphi_\infty(x)^\top \varphi_\infty(y) &= \sum_{i=0}^{\infty} \frac{\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)}{\sqrt{i!}\sqrt{i!}} \frac{(xy)^i}{\sigma^{2i}} && (9)\\
&= \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \sum_{i=0}^{\infty} \frac{(xy)^i}{i!\sigma^{2i}} && (10)\\
&= \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \sum_{i=0}^{\infty} \frac{1}{i!} \left(\frac{xy}{\sigma^2}\right)^i && (11)\\
&\underset{eq.8}{=} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \exp\left(\frac{xy}{\sigma^2}\right) && (12)\\
&= \exp\left(-\frac{(x-y)^2}{2\sigma^2}\right), && (13)
\end{aligned}$$

and find a closed form solution for the dot-product of two feature-transformed vectors.

3.3 Problem 6

Yes. For very shallow Gaussian's ($\sigma \to 0$) each point in the original data space is transformed onto an own dedicated axis. Imagine in the feature transformed space there exists one axis that shows how $(0, 0)$-ish a point is, how $(0, 0.1)$-ish a point is and so on. Here the example is shown in 2D but it is valid for any finite dimensional space. Then the data becomes linearly separable. Ultimately, if σ can be chosen arbitrarily, the choice of $\sigma \to 0$ results in a linearly separable feature space for every input.

4 Kernelized k-nearest Neighbours

4.1 Problem 7

$$\begin{aligned}
||\varphi(\mathbf{x}) - \varphi(\mathbf{y})|| &= \sqrt{(\varphi(\mathbf{x}) - \varphi(\mathbf{y}))^\top (\varphi(\mathbf{x}) - \varphi(\mathbf{y}))} && (14)\\
&= \sqrt{\varphi^\top(\mathbf{x})\varphi(\mathbf{x}) - 2\varphi^\top(\mathbf{x})\varphi(\mathbf{y}) + \varphi^\top(\mathbf{y})\varphi(\mathbf{y})} && (15)\\
&= \sqrt{K(\mathbf{x},\mathbf{x}) - 2K(\mathbf{x},\mathbf{y}) + K(\mathbf{y},\mathbf{y})}\,. && (16)
\end{aligned}$$

Machine Learning Worksheet 09

Deep Learning

1 Activation functions

Problem 1: Why do we use basis functions in neural networks? What purpose do they serve?

Problem 2: Consider a neural network with hidden-unit nonlinear sigmoid activation functions. Show that there exists an equivalent network, which computes exactly the same function, but with hidden unit activation functions given by $\tanh(x)$.

Problem 3: We already know that the derivative of the sigmoid activation function can be expressed in terms of the function value itself. Show that the derivative of the tanh activation function can also be expressed in terms of the function value itself. Why is this a useful property?

Problem 4: The error function for binary classification problems,

$$E(\boldsymbol{W}) = -\sum_{i=1}^{N}\Big(y_i \log f(\boldsymbol{x}_i, \boldsymbol{W}) + (1-y_i)\log\big[1 - f(\boldsymbol{x}_i, \boldsymbol{W})\big]\Big)$$

is derived for a network having a logistic sigmoid activation function so that $0 \leq f(\boldsymbol{x}_i, \boldsymbol{W}) \leq 1$, and data have target values $y_i \in \{0, 1\}$. Derive the corresponding error function if we consider a network having an output $-1 \leq f(\boldsymbol{x}_i, \boldsymbol{W}) \leq 1$ and target values $y_i \in \{-1, 1\}$. What would be the appropriate choice of activation function in this case?

2 Optimization

Problem 5: A simple neural network has as loss function

$$E(\boldsymbol{w}) := \frac{1}{m}\sum_{i=1}^{m} \ell(y_i - \boldsymbol{w}\cdot\boldsymbol{x}_i) + \lambda\|\boldsymbol{w}\|^2/2$$

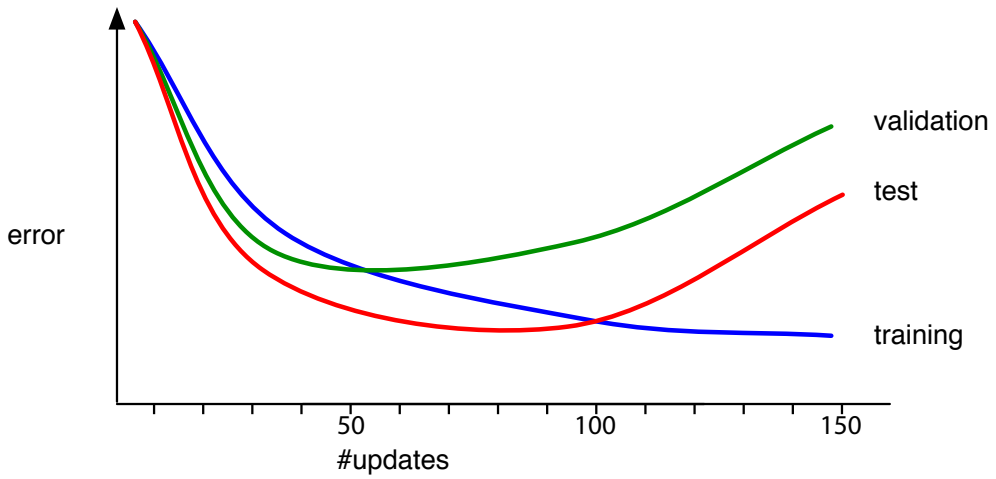
where

$$\ell(\eta) = \begin{cases} \frac{1}{2}\eta^2 & \text{if } |\eta| < 1 \\ |\eta| - \frac{1}{2} & \text{otherwise.} \end{cases}$$

Here, $\boldsymbol{x}_i \in \mathbb{R}^D$ are the data, $\boldsymbol{w} \in \mathbb{R}^D$ are the weights, and $y_i \in \mathbb{R}$ the target outputs for data points $i = 1, 2, \ldots, m$. The λ is a constant.

Compute the gradient of $E(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$, when optimising over all data.

Problem 6: When do you "stop training" (give the approximate number of the update/iteration) and use the corresponding neural network weights? Explain your answer. Relate your answer to the figure below in which the data is separated in a training set, a validation set, and a test set.



3 Numerical stability

Problem 7: In machine learning you quite often come across problems which contain the following quantity

$$y = \log \sum_{i=1}^{N} e^{x_i}$$

For example if we want to calculate the log-likelihood of neural network with a softmax output we get this quantity due to the normalization constant. If you try to calculate it naively, you will quickly encounter underflows or overflows, depending on the scale of x_i. Despite working in log-space, the limited precision of computers is not enough and the result will be ∞ or $-\infty$.

To combat this issue we typically use the following identity:

$$y = \log \sum_{i=1}^{N} e^{x_i} = a + \log \sum_{i=1}^{N} e^{x_i - a}$$

for an arbitrary a. This means, you can shift the center of the exponential sum. A typical value is setting a to the maximum ($a = \max_i x_i$), which forces the greatest value to be zero and even if the other values would underflow, you get a reasonable result.

Your task is to show that the identity holds.

Problem 8: Similar to the previous exercise we can compute the output of the softmax function $\pi_i = \frac{e^{x_i}}{\sum_{i=1}^{N} e^{x_i}}$ in a numerically stable way by introducing an arbitrary a:

$$\frac{e^{x_i}}{\sum_{i=1}^{N} e^{x_i}} = \frac{e^{x_i - a}}{\sum_{i=1}^{N} e^{x_i - a}}$$

often chosen $a = \max_i x_i$. Show that the above identity holds.

Problem 9: Let the logits (before applying sigmoid) of a single training example be x and the corresponding label be y. The sigmoid logistic loss (i.e. binary cross-entropy) for this example is then:

$$-(y \log(\sigma(x)) + (1 - y) \log(1 - \sigma(x)))$$

To ensure stability and avoid overflow, usually implementations use this equivalent formulation:

$$\max(x, 0) - x \cdot y + \log(1 + e^{-\text{abs}(x)})$$

You task is to show that the equivalence holds.

Homework 09 - Deep Learning

IN2064 Machine Learning

Arne Sachtler - *Registration Number: 03692662*

January 8, 2018

1 Activation Functions

1.1 Problem 1

Basis function make intrinsically linearly inseparable data separable. Additionally, without activation functions the concatenation of multiple neurons would make no sense as all linear operation could than be combined to one single linear operator.

1.2 Problem 2

Consider a neuron of the hidden layers in the tanh-network and the sigmoid-network, respectively. Each neuron has a affine operation to the input variables and an activation function. Using the identity $e^{-x} = e^{-\frac{1}{2}x-\frac{1}{2}x} = \frac{\exp(-\frac{1}{2}x)}{\exp(\frac{1}{2}x)}$ and starting with the sigmoid function we can express sigmoid in terms of tanh

$$\begin{aligned}
\sigma(x) &= \frac{1}{1+e^{-x}} && (1)\\
&= \frac{e^{\frac{1}{2}x}}{e^{\frac{1}{2}x}+e^{-\frac{1}{2}x}} && (2)\\
&= \frac{1}{2}\frac{2e^{\frac{1}{2}x}}{e^{\frac{1}{2}x}+e^{-\frac{1}{2}x}} && (3)\\
&= \frac{1}{2}\left(\frac{e^{\frac{1}{2}x}+e^{-\frac{1}{2}x}}{e^{\frac{1}{2}x}+e^{-\frac{1}{2}x}}\right) && (4)\\
&= \frac{1}{2}\left(\tanh\frac{x}{2}+1\right) && (5)
\end{aligned}$$

Now again consider a neurons in the neural network with tanh activation functions. If the affine mapping in the neurons scales the input by $\frac{1}{2}$ and adds a bias of 1 the neuron models a sigmoid function. The next neurons again weight the inputs of the previous layer and add biases accordingly.

1.3 Problem 3

Derivative of tanh:

$$\begin{aligned}
\frac{d}{dx}\tanh(x) &= \frac{d}{dx}\frac{e^x - e^{-x}}{e^x + e^{-x}} && (6)\\
&= \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-}x)}{(e^x + e^{-x})^2} && (7)\\
&= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} && (8)\\
&= 1 - \tanh^2(x) && (9)
\end{aligned}$$

The property of the function itself appearing in the derivative is useful as previously computed values can be reused.

1.4 Problem 4

The tanh should be used as its range is from -1 to 1. The loss function can be computed by

$$E(\mathbf{W}) = \sum_{i=1}^{N} (\mathbb{1}(y_1 = 1)(1 - f(x_i, \mathbf{W})) + \mathbb{1}(y_i = -1)(f(x_i, \mathbf{W}) + 1)) \tag{10}$$

2 Optimization

2.1 Problem 5

For the derivative of the loss we get

$$\frac{dE}{d\mathbf{w}} = \frac{1}{m}\sum_i -\mathbf{x}_i \cdot l'(y_i - \mathbf{w}^\top \mathbf{x}_i) + \lambda||\mathbf{w}|| \tag{11}$$

where

$$l'(\eta) = \begin{cases} 1 & \text{if } \eta \geq 1, \\ -1 & \text{if } \eta \leq -1, \\ \eta & \text{otherwise.} \end{cases} \tag{12}$$

2.2 Problem 6

I would stop training as soon as the validation error starts to rise again. In the example is approximately at the 50th iteration.

3 Numerical Instability

3.1 Problem 7

$$
\begin{aligned}
a + \log \sum_i e^{x_i - a} &= a + \log \sum_i \frac{e^{x_i}}{e^a} && (13)\\
&= a + \log \left(\frac{1}{e^a} \sum_i e^{x_i} \right) && (14)\\
&= a + \log \frac{1}{e^a} + \log \sum_i e^{x_i} && (15)\\
&= \log \sum_i e^{x_i} \,. && (16)
\end{aligned}
$$

■

3.2 Problem 8

In both the numerator and the denominator the factor $\frac{1}{e^a}$ can be factorized out and the fraction can be reduced. In the denominator the factor $\frac{1}{e^a}$ can be factorized out as it is a constant factor in the sum. ■

3.3 Problem 9

First the logarithmic sigmoid functions are computed. We get

$$\log \sigma(x) = \log \frac{1}{1 + e^{-x}} = -\log(1 + e^{-x}) \tag{17}$$

and

$$\log(1 - \sigma(x)) = \log \frac{e^{-x}}{1 + e^{-x}} = -x - \log(1 + e^{-x}) \,. \tag{18}$$

Afterwards the binary cross entropy can be reformulated straightforwardly

$$
\begin{aligned}
& -(y \log(\sigma(x)) + (1 - y) \log(1 - \sigma(x))) && (19)\\
= \;& y \log(1 + e^{-x}) - (1 - y)(-x - \log(1 + e^{-x})) && (20)\\
= \;& y \log(1 + e^{-x}) + x + \log(1 + e^{-x}) - xy - y \log(1 + e^{-x}) && (21)\\
= \;& x - xy + \log(1 + e^{-x}) \,. && (22)
\end{aligned}
$$

The latter result is fine for $x \geq 0$. For negative x the result may get unstable as e^{-x} for $x << 0$ goes to high numbers. Using the identity $1 = e^x e^{-x}$ the result can be modified to avoid the unstable exponential. Starting with the last expression we get

$$
\begin{aligned}
& x + \log(1 + e^{-x}) - xy && (23)\\
= \;& x - xy + \log(e^x e^{-x} + e^{-x}) && (24)\\
= \;& x - xy + \log(e^{-x}(e^x + 1)) && (25)\\
= \;& -xy + \log(1 + e^x) \,. && (26)
\end{aligned}
$$

Combining these we get

$$
\begin{aligned}
& -(y \log(\sigma(x)) + (1-y) \log(1-\sigma(x))) && (27)\\
= & \begin{cases} x - xy + \log(1+e^{-x}) & \text{if } x \geq 0, \\ -xy + \log(1+e^{x}) & \text{if } x < 0. \end{cases} && (28)\\
= & \max(0,x) - xy + \log(1+e^{-|x|})\,. && (29)
\end{aligned}
$$

■

Machine Learning Worksheet 10

Dimensionality Reduction

Problem 1: In this exercise, we use proof by induction to show that the linear projection onto an M-dimensional subspace that maximizes the variance of the projected data is defined by the M eigenvectors of the data covariance matrix S, given by

$$S = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^T \qquad \bar{\boldsymbol{x}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_n$$

corresponding to the M largest eigenvalues. In Section 12.1 in Bishop this result was proven for the case of $M = 1$. Now suppose the result holds for some general value of M and show that it consequently holds for dimensionality $M + 1$. To do this, first set the derivative of the variance of the projected data with respect to a vector u_{M+1} defining the new direction in data space equal to zero. This should be done subject to the constraints that u_{M+1} be orthogonal to the existing vectors $u_1, \dots u_M$, and also that it be normalized to unit length. Use Lagrange multipliers to enforce these constraints. Then make use of the orthonormality properties of the vectors $u_1, \dots, u_M$ to show that the new vector u_{M+1} is an eigenvector of S. Finally, show that the variance is maximized if the eigenvector is chosen to be the one corresponding to eigenvector λ_{M+1} where the eigenvalues have been ordered in decreasing value.

Problem 2: Consider the latent space distribution

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, \boldsymbol{I})$$

and a conditional distribution for the observed variable $\boldsymbol{x} \in \mathbb{R}^d$,

$$p(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \boldsymbol{\Phi})$$

where $\boldsymbol{\Phi}$ is an arbitrary symmetric, positive-definite noise covariance variable. Now suppose that we make a nonsingular linear transformation of the data variables $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ where $\boldsymbol{A}$ is a non-singular $d \times d$ matrix. If $\boldsymbol{\mu}_{ML}$, $\boldsymbol{W}_{ML}$, and $\boldsymbol{\Phi}_{ML}$ represent the maximum likelihood solution corresponding to the original untransformed data, show that $\boldsymbol{A}\boldsymbol{\mu}_{ML}$, $\boldsymbol{A}\boldsymbol{W}_{ML}$, and $\boldsymbol{A}\boldsymbol{\Phi}_{ML}\boldsymbol{A}^T$ will represent the corresponding maximum likelihood solution for the transformed data set. Finally, show that the form of the model is preserved if $\boldsymbol{A}$ is orthogonal and $\boldsymbol{\Phi}$ is proportional to the unit matrix so $\boldsymbol{\Phi} = \sigma^2\boldsymbol{I}$ (i.e. probabilistic PCA). The transformed $\boldsymbol{\Phi}$ matrix remains proportional to the unit matrix, and hence probabilistic PCA is covariant under a rotation of the axes of data space, as is the case for conventional PCA.

Problem 3: Use the SVD shown below. Suppose a new user Leslie assigns rating 3 to Alien and rating 4 to Titanic, giving us a representation of Leslie in the 'original space' of $[0, 3, 0, 0, 4]$. Find the representation of Leslie in concept space. What does that representation predict about how well Leslie would like the other movies appearing in our example data?

Upload a single PDF file with your solution to Moodle by 15.01.2018, 9:59am CET. The solution must be typeset (using Latex, Word, etc.). Handwritten solutions are not accepted.

	Matrix	Alien	Star Wars	Casablanca	Titanic
Joe	1	1	1	0	0
Jim	3	3	3	0	0
John	4	4	4	0	0
Jack	5	5	5	0	0
Jill	0	0	0	4	4
Jenny	0	0	0	5	5
Jane	0	0	0	2	2

Figure 11.6: Ratings of movies by users

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} .14 & 0 \\ .42 & 0 \\ .56 & 0 \\ .70 & 0 \\ 0 & .60 \\ 0 & .75 \\ 0 & .30 \end{bmatrix} \begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix} \begin{bmatrix} .58 & .58 & .58 & 0 & 0 \\ 0 & 0 & 0 & .71 & .71 \end{bmatrix}$$

$$M \qquad U \qquad \Sigma \qquad V^{\mathrm{T}}$$

Problem 4: Load the notebook `10_homework_dim_reduction.ipynb` from Piazza. Fill in the missing code and run the notebook. Convert the evaluated notebook to pdf and add it to the printout of your homework.

Note: We suggest that you use Anaconda for installing Python and Jupyter, as well as for managing packages. We recommend that you use Python 3.

For more information on Jupyter notebooks and how to convert them to other formats, consult the Jupyter documentation and nbconvert documentation.

Homework 10 - Dimensionality Reduction

IN2064 Machine Learning

Arne Sachtler - *Registration Number: 03692662*

January 14, 2018

1 Problem 1

First we use the Langrangian formalism in order to compute our new vector u_{M+1}

$$u_{M+1} = \arg\max_u \left(L(u, \lambda_0, \ldots, \lambda_M)\right) = \arg\max_u \left(u^\top S u + \lambda_0 (1 - u^\top u) + \sum_{i=1}^{M} \lambda_i u_i^\top u \right) . \tag{1}$$

Computing the derivative we obtain

$$\left(\frac{\partial L}{\partial u}\right)^\top = 2u^\top S - 2\lambda_0 u^\top + \sum_{i=1}^{M} \lambda_i u_i^\top \stackrel{!}{=} 0 \tag{2}$$

Let's consider left multiply the equation with the last eigenvector u_M and u will be called u_{M+1} for consistency reasons. This multiplication is allowed as u_M is always non-zero.

$$\begin{aligned}
0 &= u_M^\top \left(2Su_{M+1} - 2\lambda_0 u_{M+1} + \sum_{i=1}^{M} \lambda_i u_i \right) && (3)\\
&= 2u_M^\top S u_{M+1} - 2\lambda_0 u_M^\top u_{M+1} + \sum_{i=1}^{M} \lambda_i u_M^\top u_i && (4)\\
&= u_M^\top S u_{M+1} - \lambda_0 u_M^\top u_{M+1} && (5)\\
&= S u_{M+1} - \lambda_0 u_{M+1} && (6)
\end{aligned}$$

Here the orthogonality property

$$\forall i, j \in \{1 \ldots M\} : i \neq j \implies u_i^\top u_j = 0 \tag{7}$$

was used. Finally we get the eigenvalue problem

$$S u_{M+1} = \lambda_0 u_{M+1} \, . \tag{8}$$

■

2 Problem 2

First let us recap a fundamental rule for linear transformation of Gaussian random variables. Given a normal-distributed random variable X with mean $\boldsymbol{\mu}_X$ and variance $\boldsymbol{\Sigma}_X$, more formally

$$X \sim \mathcal{N}(\boldsymbol{\mu}_X, \boldsymbol{\Sigma}_X)\,, \tag{9}$$

we can construct a new normal-distributed random variable Y using the linear transformation $\mathbf{y} = \mathbf{A}\mathbf{x}$. Then, the constructed random variable Y is distributed as

$$Y \sim \mathcal{N}(\boldsymbol{\mu}_Y, \boldsymbol{\Sigma}_Y) = \mathcal{N}(\mathbf{A}\boldsymbol{\mu}_X, \mathbf{A}^\top \boldsymbol{\Sigma}_X \mathbf{A})\,. \tag{10}$$

It follows that

$$E[Y] = \boldsymbol{\mu}_Y = \mathbf{A}\boldsymbol{\mu}_X \tag{11}$$

and

$$Var[Y] = \boldsymbol{\Sigma}_Y = \mathbf{A}\boldsymbol{\Sigma}_X\mathbf{A}^\top\,. \tag{12}$$

Given the maximum likelihood estimates $\boldsymbol{\mu}_{ML}$, $\boldsymbol{W}_{ML}$ and $\boldsymbol{\Phi}_{ML}$ we know the data X is distributed as

$$X \sim \mathcal{N}(\boldsymbol{\mu}_{ML}, \mathbf{M}_{ML}\mathbf{M}_{ML}^\top + \boldsymbol{\Phi}_{ML})\,. \tag{13}$$

Now let Y be the random variable of the linearly transformed data with the transformation rule $\mathbf{y} = \mathbf{A}\mathbf{x}$. Using the rules for linear transformation of Gaussian random variables we get for the expected value

$$E[Y] = \mathbf{A}\boldsymbol{\mu}_{ML} \tag{14}$$

and for the variance

$$\begin{aligned} Var[Y] &= \mathbf{A}\left(\mathbf{M}_{ML}\mathbf{M}_{ML}^\top + \boldsymbol{\Phi}_{ML}\right)\mathbf{A}^\top && (15)\\ &= \mathbf{A}\mathbf{W}_{ML}\mathbf{W}_{ML}^\top\mathbf{A}^\top + \mathbf{A}\boldsymbol{\Phi}_{ML}\mathbf{A}^\top && (16)\\ &= \mathbf{A}\mathbf{W}_{ML}\left(\mathbf{A}\mathbf{W}_{ML}\right)^\top + \mathbf{A}\boldsymbol{\Phi}_{ML}\mathbf{A}^\top\,. && (17) \end{aligned}$$

Comparing this to the PPCA formulation we see that the new maximum likelihood estimates for the transformed data space are

$$\begin{aligned} \boldsymbol{\mu}'_{ML} &= \mathbf{A}\boldsymbol{\mu}_{ML} && (18)\\ \boldsymbol{W}'_{ML} &= \mathbf{A}\boldsymbol{W}_{ML} && (19)\\ \boldsymbol{\Phi}'_{ML} &= \mathbf{A}\boldsymbol{\Phi}_{ML}\mathbf{A}^\top\,. && (20) \end{aligned}$$

Finally, given that $\boldsymbol{\Phi} = \sigma^2\mathbf{I}$ we apply the transformation A:

$$\boldsymbol{\Phi}' = \mathbf{A}\boldsymbol{\Phi}\mathbf{A}^\top = \sigma^2\mathbf{A}\mathbf{I}\mathbf{A}^\top = \sigma^2\mathbf{A}\mathbf{A}^\top = \sigma^2\mathbf{I}\,, \tag{21}$$

as $\mathbf{A}$ is orthogonal. ■

3 Problem 3

Leslie in concept space is

$$\begin{pmatrix} 0 & 3 & 0 & 0 & 4 \end{pmatrix} \mathbf{V} = \begin{pmatrix} 1.74 & 2.84 \end{pmatrix} . \tag{22}$$

Interpretation: Leslie likes everything, but has a tendency to romantic movies.

4 Problem 4

See the pages attached.

10_homework_dim_reduction

January 14, 2018

1 Programming assignment 10: Dimensionality Reduction

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

        %matplotlib inline
```

1.1 PCA Task

Given the data in the matrix X your tasks is to: * Calculate the covariance matrix Σ. * Calculate eigenvalues and eigenvectors of Σ. * Plot the original data X and the eigenvectors to a single diagram. What do you observe? Which eigenvector corresponds to the smallest eigenvalue? * Determine the smallest eigenvalue and remove its corresponding eigenvector. The remaining eigenvector is the basis of a new subspace. * Transform all vectors in X in this new subspace by expressing all vectors in X in this new basis.

1.1.1 The given data X

```
In [2]: X = np.array([(-3,-2),(-2,-1),(-1,0),(0,1),
                      (1,2),(2,3),(-2,-2),(-1,-1),
                      (0,0),(1,1),(2,2), (-2,-3),
                      (-1,-2),(0,-1),(1,0), (2,1),(3,2)])
```

1.1.2 Task 1: Calculate the covariance matrix Σ

```
In [3]: def get_covariance(X):
            """Calculates the covariance matrix of the input data.

            Parameters
            ----------
            X : array, shape [N, D]
                Data matrix.

            Returns
            -------
            Sigma : array, shape [D, D]
                Covariance matrix
```

```
    """
    xmean = np.sum(X, axis=0)/X.shape[0]
    return X.T.dot(X) - np.outer(xmean,xmean)
```

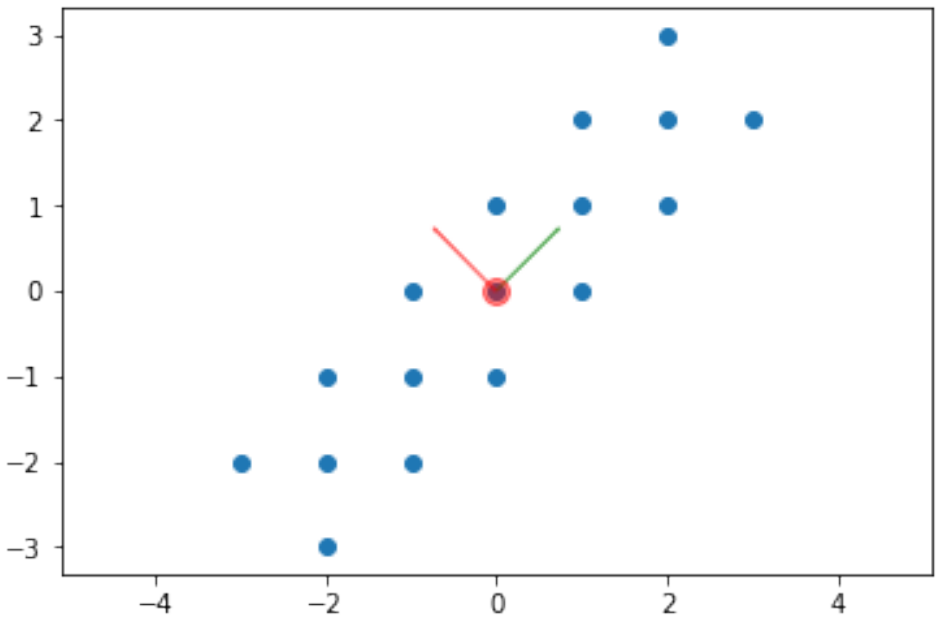
1.1.3 Task 2: Calculate eigenvalues and eigenvectors of Σ.

```
In [4]: def get_eigen(S):
    """Calculates the eigenvalues and eigenvectors of the input matrix.

    Parameters
    ----------
    S : array, shape [D, D]
        Square symmetric positive definite matrix.

    Returns
    -------
    L : array, shape [D]
        Eigenvalues of S
    U : array, shape [D, D]
        Eigenvectors of S

    """
    L, U = np.linalg.eig(S)
    return L, U.T
```

1.1.4 Task 3: Plot the original data X and the eigenvectors to a single diagram.

```
In [5]: # plot the original data
    plt.scatter(X[:, 0], X[:, 1])

    # plot the mean of the data
    mean_d1, mean_d2 = X.mean(0)
    plt.plot(mean_d1, mean_d2, 'o', markersize=10, color='red', alpha=0.5)

    # calculate the covariance matrix
    Sigma = get_covariance(X)
    # calculate the eigenvector and eigenvalues of Sigma
    L, U = get_eigen(Sigma)

    plt.axis('equal')
    plt.arrow(mean_d1, mean_d2, U[0, 0], U[0, 1], width=0.01, color='green', alpha=0.5)
    plt.arrow(mean_d1, mean_d2, U[1, 0], U[1, 1], width=0.01, color='red', alpha=0.5)
```

```
Out[5]: <matplotlib.patches.FancyArrow at 0x7f9b8ce11da0>
```



What do you observe in the above plot? Which eigenvector corresponds to the smallest eigenvalue?

Write your answer here:

$\left(-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$

1.1.5 Task 4: Transform the data

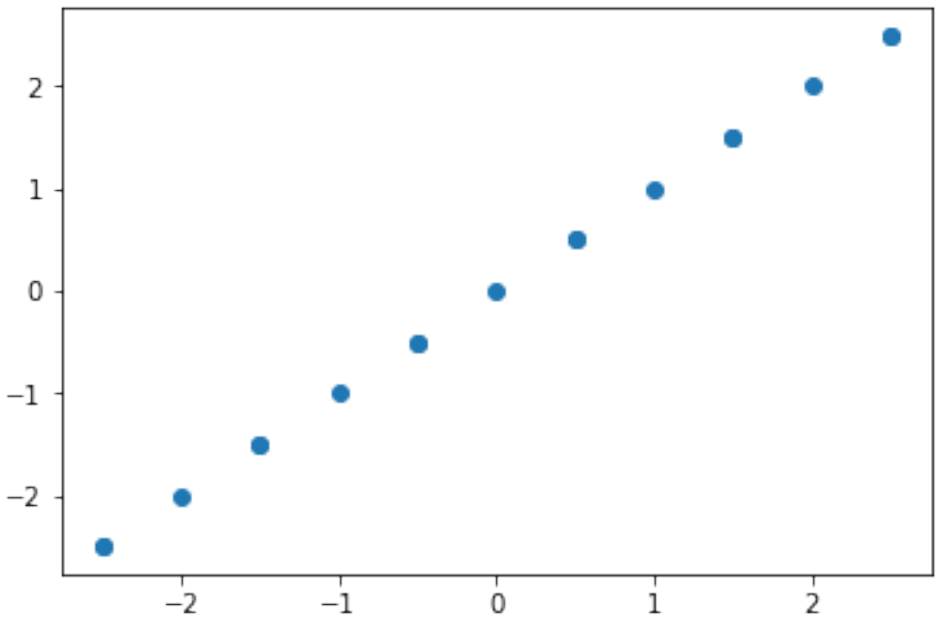
Determine the smallest eigenvalue and remove its corresponding eigenvector. The remaining eigenvector is the basis of a new subspace. Transform all vectors in X in this new subspace by expressing all vectors in X in this new basis.

```
In [6]: def transform(X, U, L):
            """Transforms the data in the new subspace spanned by the eigenvector correspondin
            Parameters
            ----------
            X : array, shape [N, D]
                Data matrix.
            L : array, shape [D]
                Eigenvalues of Sigma_X
            U : array, shape [D, D]
                Eigenvectors of Sigma_X

            Returns
            -------
            X_t : array, shape [N, 1]
                Transformed data
```

```
    """
    Xtilde = X - np.sum(X, axis=0)/X.shape[0]
    g = U[np.argmax(L), :]
    return Xtilde.dot(g)
```

In [7]:
```
X_t = transform(X, U, L)
```

In [8]:
```
# Transform the reduced points back to the original space to validate the results
v = U[np.argmax(L), :]
Xnew = np.array([y * v for y in X_t])
plt.scatter(Xnew[:,0], Xnew[:,1])
plt.show()
```



1.2 Task SVD

1.2.1 Task 5: Given the matrix M find its SVD decomposition $M = U \cdot \Sigma \cdot V$ and reduce it to one dimension using the approach described in the lecture.

In [9]:
```
M = np.array([[1, 2], [6, 3],[0, 2]])
```

In [10]:
```
def reduce_to_one_dimension(M):
    """Reduces the input matrix to one dimension using its SVD decomposition.
    Parameters
    ----------
    M : array, shape [N, D]
        Input matrix.
```

```
        Returns
        -------
        M_t: array, shape [N, 1]
            Reduce matrix.
        """
        U,S,V = np.linalg.svd(M)
        return M.dot((V.T)[0,:])
```

In [11]:
```
M_t = reduce_to_one_dimension(M)
print(M_t)
```

```
[-1.90211303 -6.68109819 -1.05146222]
```

Machine Learning Worksheet 11

Clustering

1 Gaussian Mixture Model

Problem 1: Consider a mixture of K Gaussians

$$p(\boldsymbol{x}) = \sum_k \pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Derive the expected value $\mathbb{E}[\boldsymbol{x}]$ and the covariance $\mathrm{Cov}[\boldsymbol{x}]$.

Hint: it is helpful to remember the identity $\mathrm{Cov}[\boldsymbol{x}] = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T] - \mathbb{E}[\boldsymbol{x}]\,\mathbb{E}[\boldsymbol{x}]^T$.

Problem 2: Consider a mixture of K isotropic Gaussians, all with the same *known* covariances $\boldsymbol{\Sigma}_k = \sigma^2 \boldsymbol{I}$.

Derive the EM algorithm for the case when $\sigma^2 \to 0$, and show that it's equivalent to Lloyd's algorithm for K-means.

Problem 3: Load the notebook `11_homework_clustering.ipynb` from Piazza and the dataset `faithful.txt`. Fill in the missing code and run the notebook. Convert the evaluated notebook to pdf and add it to the printout of your homework.

Note: We suggest that you use Anaconda for installing Python and Jupyter, as well as for managing packages. We recommend that you use Python 3.

For more information on Jupyter notebooks and how to convert them to other formats, consult the Jupyter documentation and nbconvert documentation.

Homework 11 - Clustering

IN2064 Machine Learning

Arne Sachtler - *Registration Number: 03692662*

January 21, 2018

1 Gaussian Mixture Model

1.1 Problem 1

Given the mixture of Gaussian

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{1}$$

the expected value can be derived straightforwardly

$$\begin{aligned} E[x] &= \int x \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \right) dx && (2) \\ &= \int \sum_{k=1}^{K} x \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) dx && (3) \\ &= \sum_{k=1}^{K} \int x \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) dx && (4) \\ &= \sum_{k=1}^{K} \pi_k \int x \mathcal{N}(x|\mu_k, \Sigma_k) dx && (5) \\ &= \sum_{k=1}^{K} \pi_k \mu_k \,. && (6) \end{aligned}$$

For the covariance we know $Cov[x] = E[xx^\top] - E[x]E[x]^\top$, so the only unknown quantity is $E[xx^\top]$. This can be computed using the theorem in the Bishop book [1, p. 82]

$$E[xx^\top] = \sum_{k=1}^{K} \pi_k (\mu_k \mu_k^\top + \Sigma_k) \,. \tag{7}$$

And finally we get

$$Cov[x] = \sum_{k=1}^{K} \pi_k(\mu_k\mu_k^\top + \Sigma_k) + \sum_{k=1}^{K} \pi_k\mu_k \sum_{k=1}^{K} \pi_k\mu_k^\top . \tag{8}$$

1.2 Problem 2

Using the isotropic Gaussians with a covariance of (in the limit) zero, the responsibilities are hard decision values. Given a data point the responsibility is implicitly set to one for that Gaussian whose mean is closest among all Gaussians of the mixture model. Afterwards in the M-step the covariances are not updated at all as they are constant and given by assumption. The means are the unweighed sample means of the data points for each cluster as in the Lloyds algorithms for k means. Finally, the N_k become natural numbers again as the responsibilities are in $\mathbb{N}$.

1.3 Problem 3

See the pdf pages attached.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

11_homework_clustering

January 21, 2018

1 Programming assignment 11: Gaussian Mixture Model

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.mlab as mlab
        import seaborn as sns
        sns.set_style('whitegrid')
        %matplotlib inline

        from scipy.stats import multivariate_normal
```

1.1 Your task

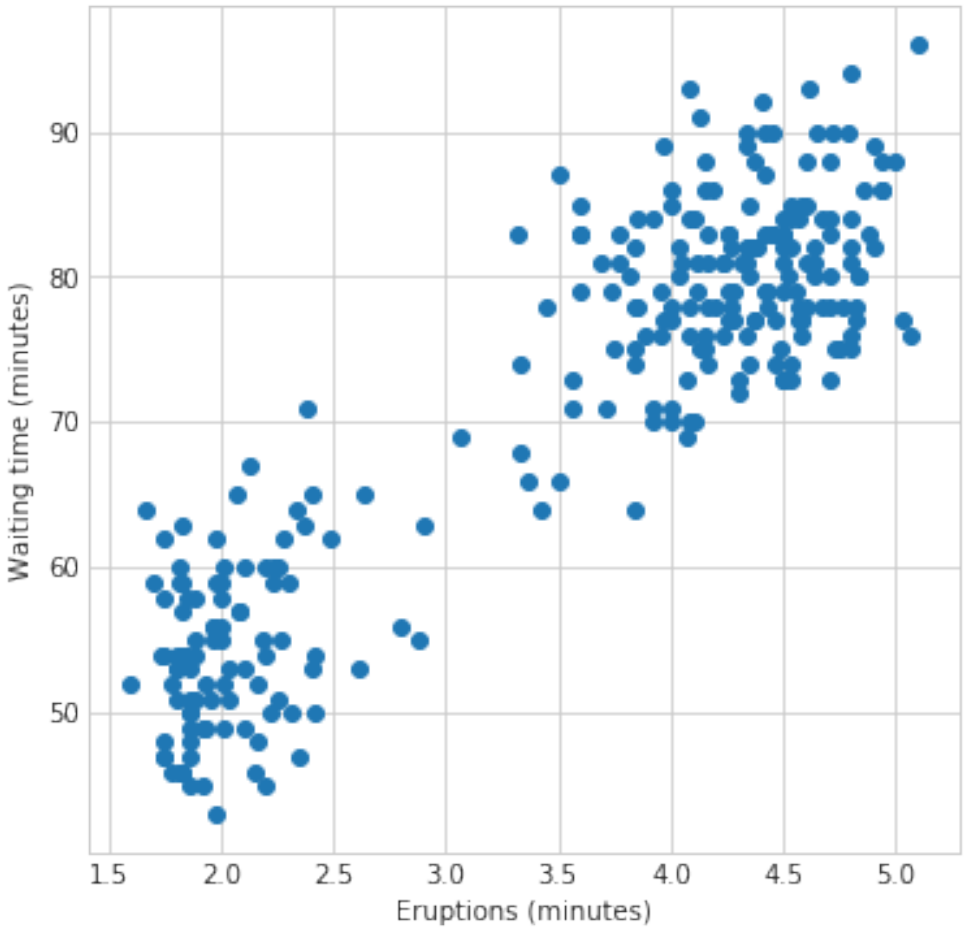
In this homework sheet we will implement Expectation-Maximization algorithm for learning & inference in a Gaussian mixture model.

We will use the dataset containing information about eruptions of a geyser called "Old Faithful". The dataset in suitable format can be downloaded from Piazza.

As usual, your task is to fill out the missing code, run the notebook, convert it to PDF and attach it you your HW solution.

1.2 Generate and visualize the data

```
In [2]: X = np.loadtxt('faithful.txt')
        plt.figure(figsize=[6, 6])
        plt.scatter(X[:, 0], X[:, 1])
        plt.xlabel('Eruptions (minutes)')
        plt.ylabel('Waiting time (minutes)')
        plt.show()
```



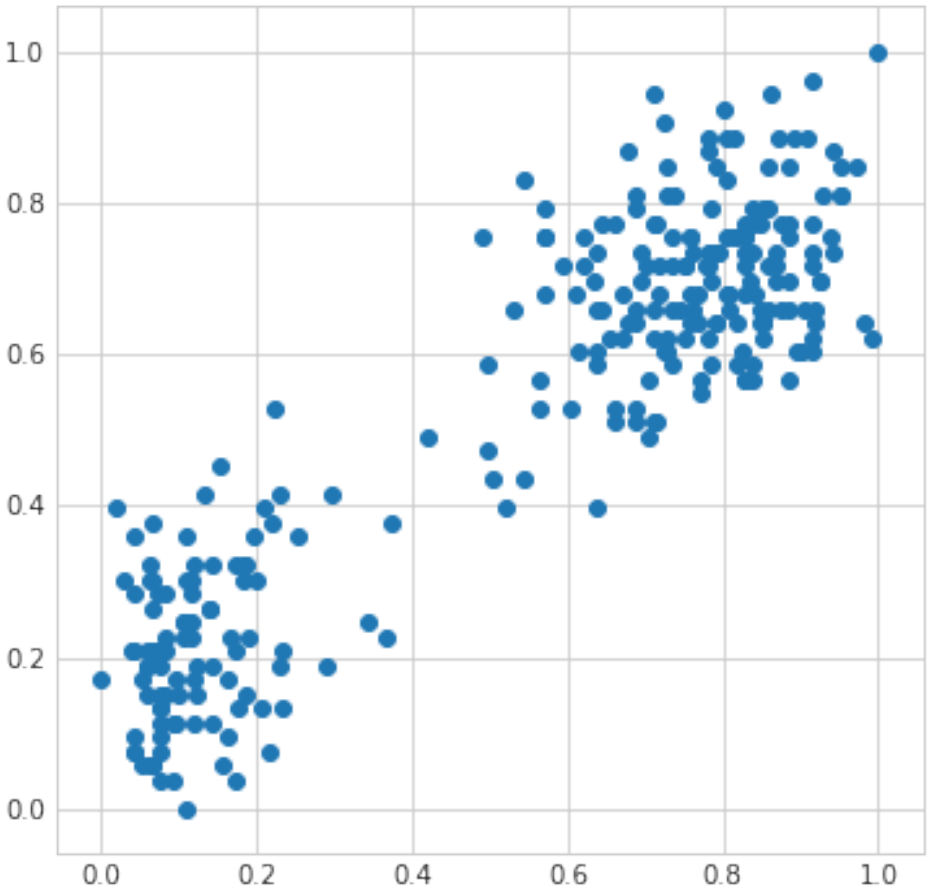
1.3 Task 1: Normalize the data

Notice, how the values on two axes are on very different scales. This might cause problems for our clustering algorithm.

Normalize the data, such that it lies in the range $[0, 1]$ along each dimension (each column of X).

```
In [11]: def normalize_data(X):
             lower = lambda a: np.amin(a, axis=0)
             higher = lambda a: np.amax(a, axis=0)
             return (X - lower(X))/(higher(X) - lower(X))
```

```
In [12]: plt.figure(figsize=[6, 6])
         X_norm = normalize_data(X)
         plt.scatter(X_norm[:, 0], X_norm[:, 1]);
```



1.4 Task 2: Compute the log-likelihood of GMM

Here and in some other places, you might want to use the function `multivariate_normal.pdf` from the `scipy.stats` package.

```
In [15]: def gmm_log_likelihood(X, means, covs, mixing_coefs):
             log_likelihood = 0.0
             for n in range(X.shape[0]):
                 acc = 0.0
                 for k in range(mixing_coefs.size):
                     acc += mixing_coefs[k] * multivariate_normal.pdf(X[n, :], means[k, :], co
                 log_likelihood += np.log(acc)
             return log_likelihood
```

1.5 Task 3: E step

```
In [19]: def e_step(X, means, covs, mixing_coefs):
             wpdf = lambda i, k: mixing_coefs[k] * multivariate_normal.pdf(X[i,:], means[k,:],
```

```
        marginalwpdf = lambda i: np.sum(np.array([wpdf(i, j) for j in range(mixing_coefs.s
        responsibilities = np.array([[wpdf(i,k)/marginalwpdf(i) for k in range(mixing_coe
        return responsibilities
```

1.6 Task 4: M step

```
In [54]: def m_step(X, responsibilities):
        N = X.shape[0]
        K = responsibilities.shape[1]
        Nk = lambda k: np.sum(responsibilities[:,k])

        def mean(k):
            l = [responsibilities[i,k]*X[i,:] for i in range(N)]
            return np.sum(np.array(l),axis=0)/Nk(k)

        means = np.array([mean(k) for k in range(K)])

        def sigma(k):
            diff = lambda i: X[i,:] - means[k,:]
            l = [responsibilities[i,k]*np.outer(diff(i),diff(i)) for i in range(N)]
            return np.sum(np.array(l),axis=0)/Nk(k)

        covs = np.array([sigma(k) for k in range(K)])

        mixing_coefs = np.array([Nk(k)/N for k in range(K)])
        return means, covs, mixing_coefs
```

1.7 Visualize the result (nothing to do here)

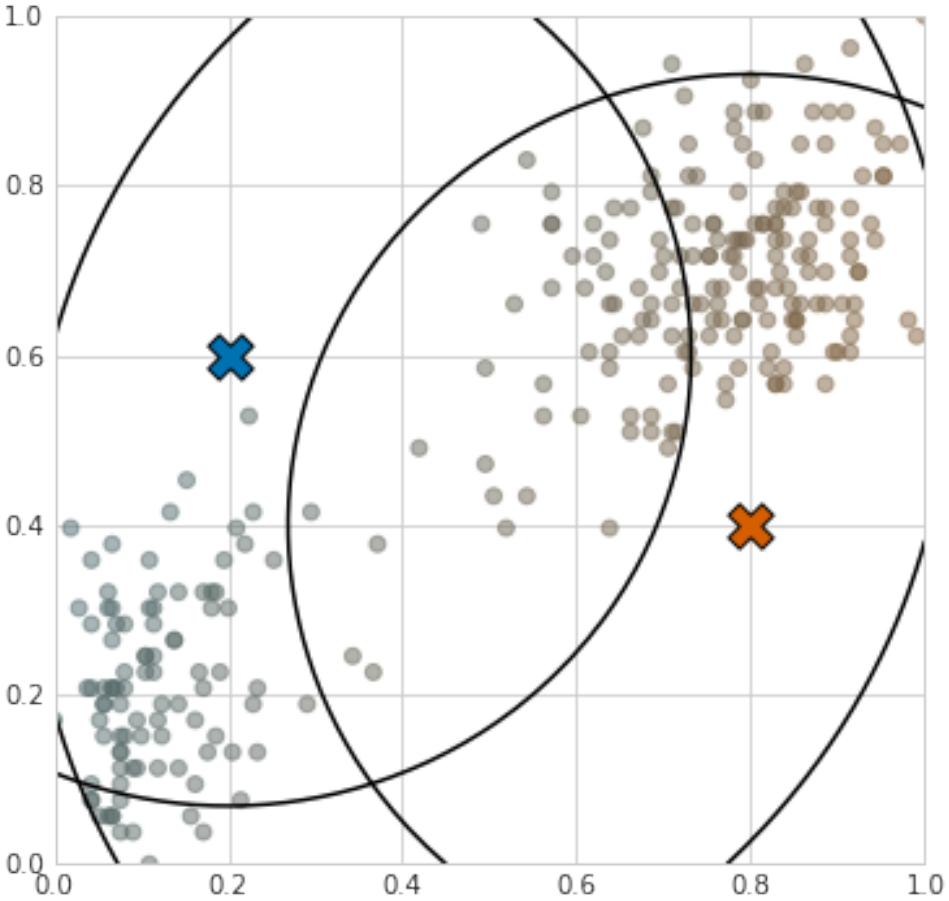
```
In [22]: def plot_gmm_2d(X, responsibilities, means, covs, mixing_coefs):
        """Visualize a mixture of 2 bivariate Gaussians.

        This is badly written code. Please don't write code like this.
        """
        plt.figure(figsize=[6, 6])
        palette = np.array(sns.color_palette('colorblind', n_colors=3))[[0, 2]]
        colors = responsibilities.dot(palette)
        # Plot the samples colored according to p(z|x)
        plt.scatter(X[:, 0], X[:, 1], c=colors, alpha=0.5)
        # Plot locations of the means
        for ix, m in enumerate(means):
            plt.scatter(m[0], m[1], s=300, marker='X', c=palette[ix],
                        edgecolors='k', linewidths=1,)
        # Plot contours of the Gaussian
        x = np.linspace(0, 1, 50)
        y = np.linspace(0, 1, 50)
        xx, yy = np.meshgrid(x, y)
        for k in range(len(mixing_coefs)):
```

```
            zz = mlab.bivariate_normal(xx, yy, np.sqrt(covs[k][0, 0]),
                                       np.sqrt(covs[k][1, 1]),
                                       means[k][0], means[k][1], covs[k][0, 1])
            plt.contour(xx, yy, zz, 2, colors='k')
        plt.xlim(0, 1)
        plt.ylim(0, 1)
        plt.show()
```

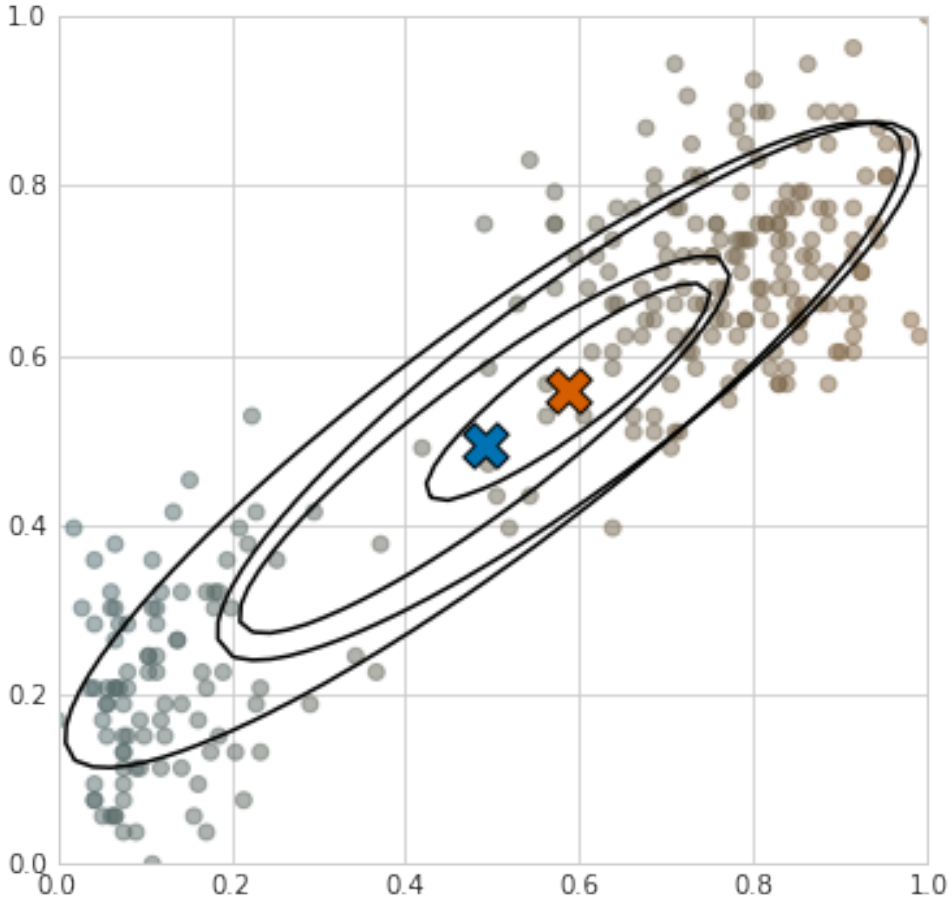
1.8 Run the EM algorithm

```
In [55]: X_norm = normalize_data(X)
         max_iters = 20

         # Initialize the parameters
         means = np.array([[0.2, 0.6], [0.8, 0.4]])
         covs = np.array([0.5 * np.eye(2), 0.5 * np.eye(2)])
         mixing_coefs = np.array([0.5, 0.5])

         old_log_likelihood = gmm_log_likelihood(X_norm, means, covs, mixing_coefs)
         print('At initialization: log-likelihood = {0}'
               .format(old_log_likelihood))
         responsibilities = e_step(X_norm, means, covs, mixing_coefs)
         plot_gmm_2d(X_norm, responsibilities, means, covs, mixing_coefs)

         # Perform the EM iteration
         for i in range(max_iters):
             responsibilities = e_step(X_norm, means, covs, mixing_coefs)
             means, covs, mixing_coefs = m_step(X_norm, responsibilities)
             new_log_likelihood = gmm_log_likelihood(X_norm, means, covs, mixing_coefs)
             # Report & visualize the optimization progress
             print('Iteration {0}: log-likelihood = {1:.2f}, improvement = {2:.2f}'
                   .format(i+1, new_log_likelihood, new_log_likelihood - old_log_likelihood))
             old_log_likelihood = new_log_likelihood
             plot_gmm_2d(X_norm, responsibilities, means, covs, mixing_coefs)
```

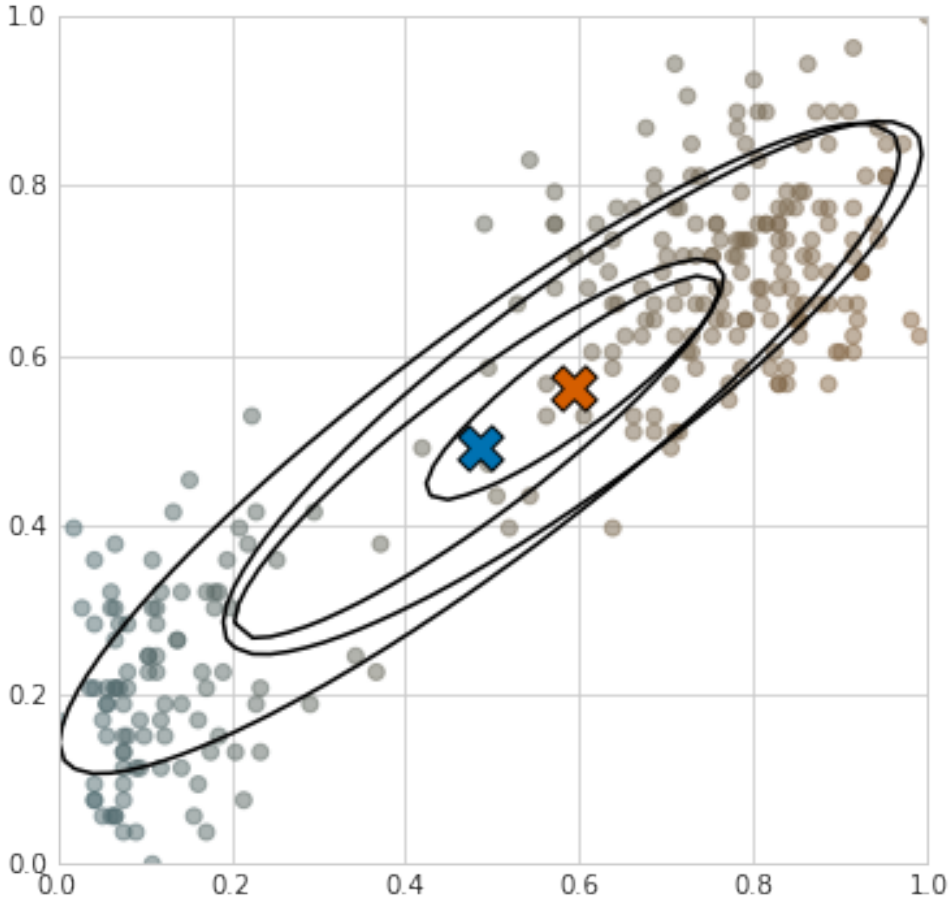
```
At initialization: log-likelihood = -382.70551524206564
```



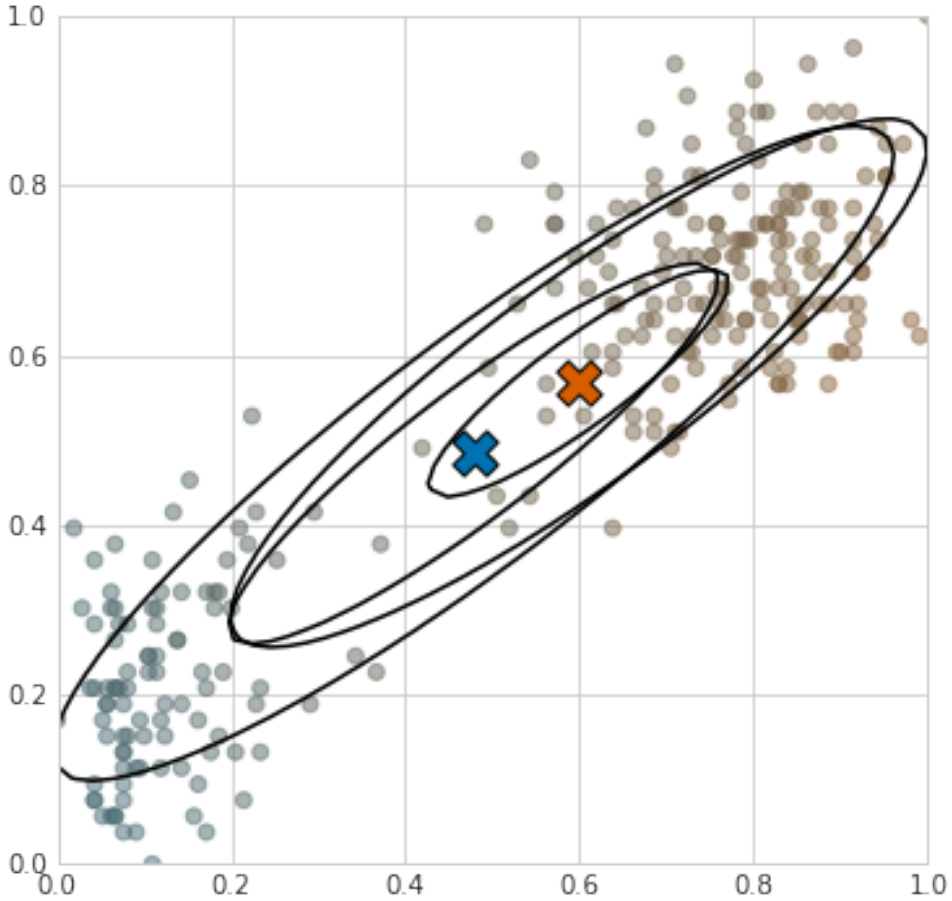

```
Iteration 1: log-likelihood = 131.29, improvement = 513.99
```



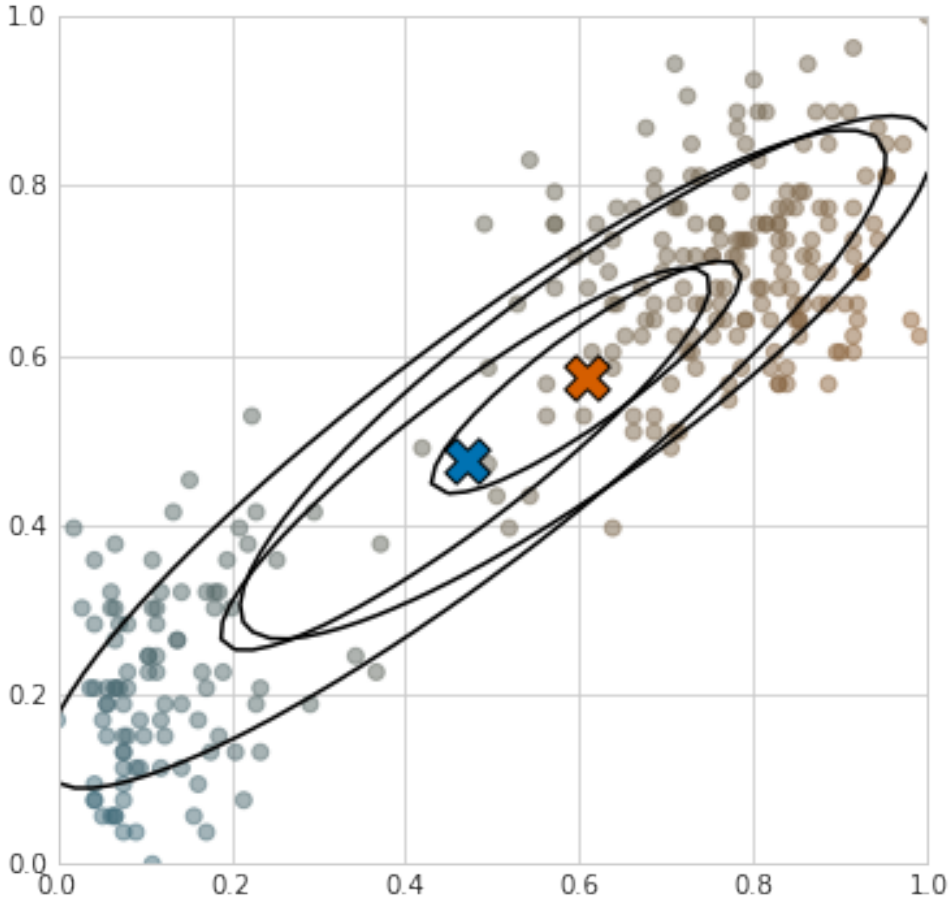

```
Iteration 2: log-likelihood = 131.48, improvement = 0.19
```



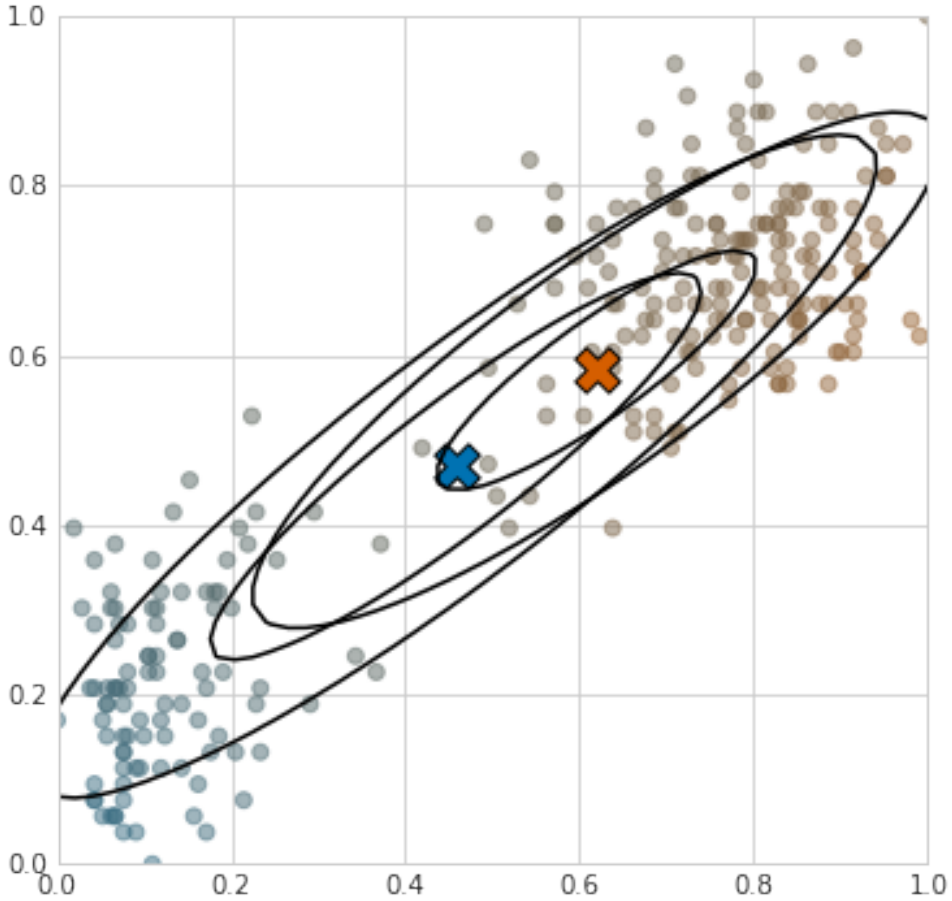

```
Iteration 3: log-likelihood = 131.75, improvement = 0.27
```



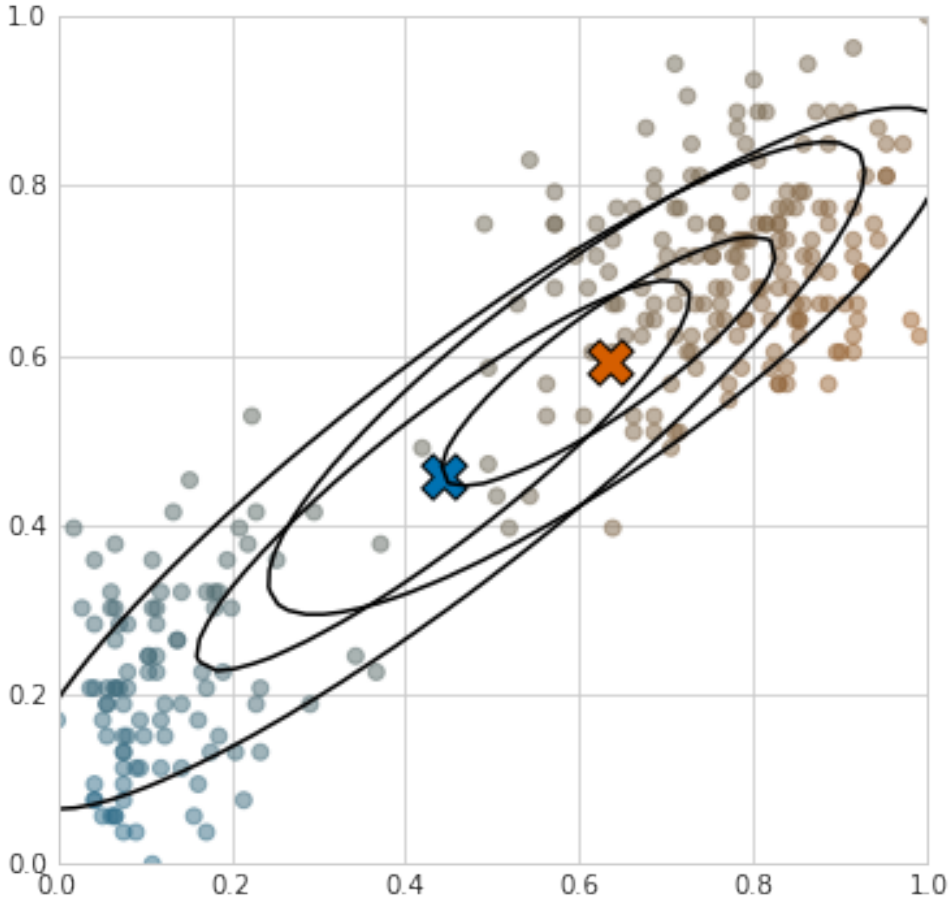

```
Iteration 4: log-likelihood = 132.15, improvement = 0.40
```



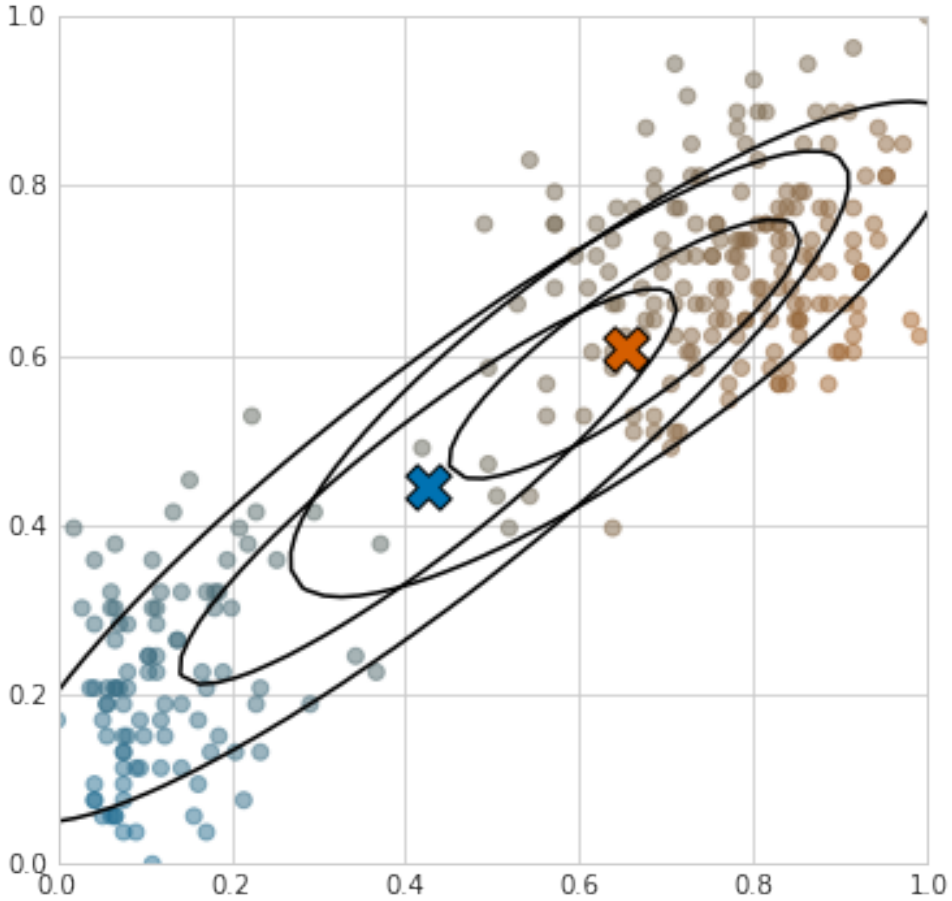

```
Iteration 5: log-likelihood = 132.77, improvement = 0.62
```



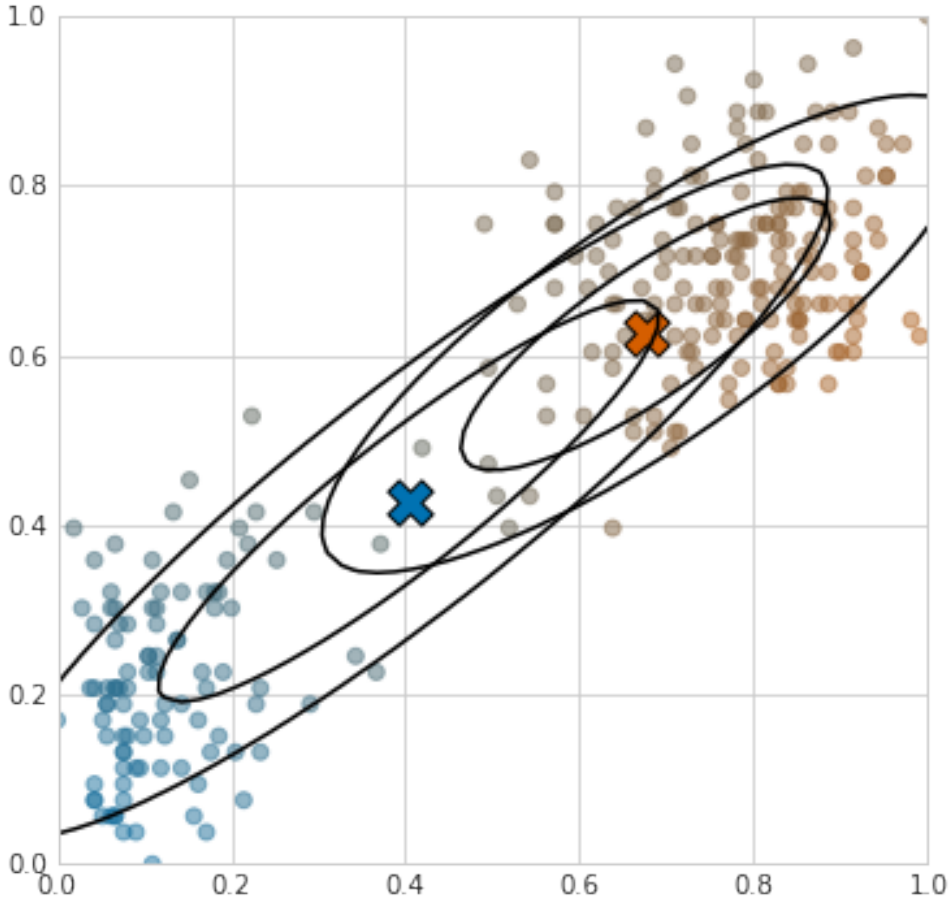

```
Iteration 6: log-likelihood = 133.81, improvement = 1.04
```



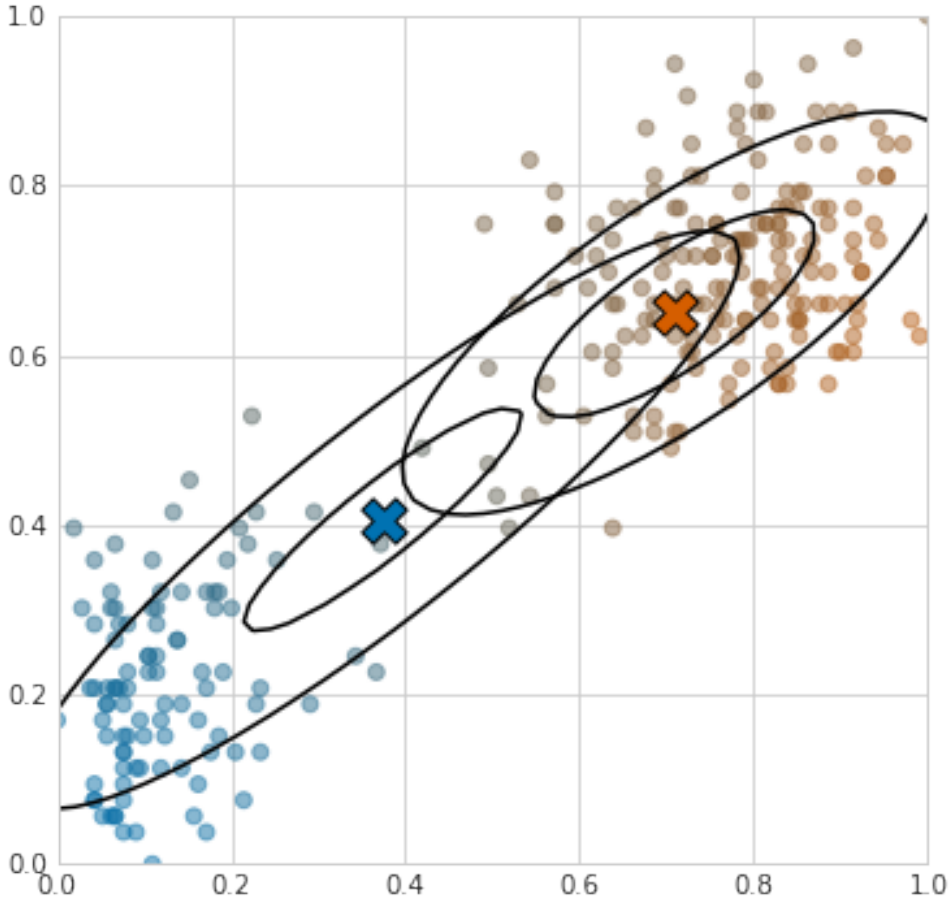

```
Iteration 7: log-likelihood = 135.74, improvement = 1.93
```



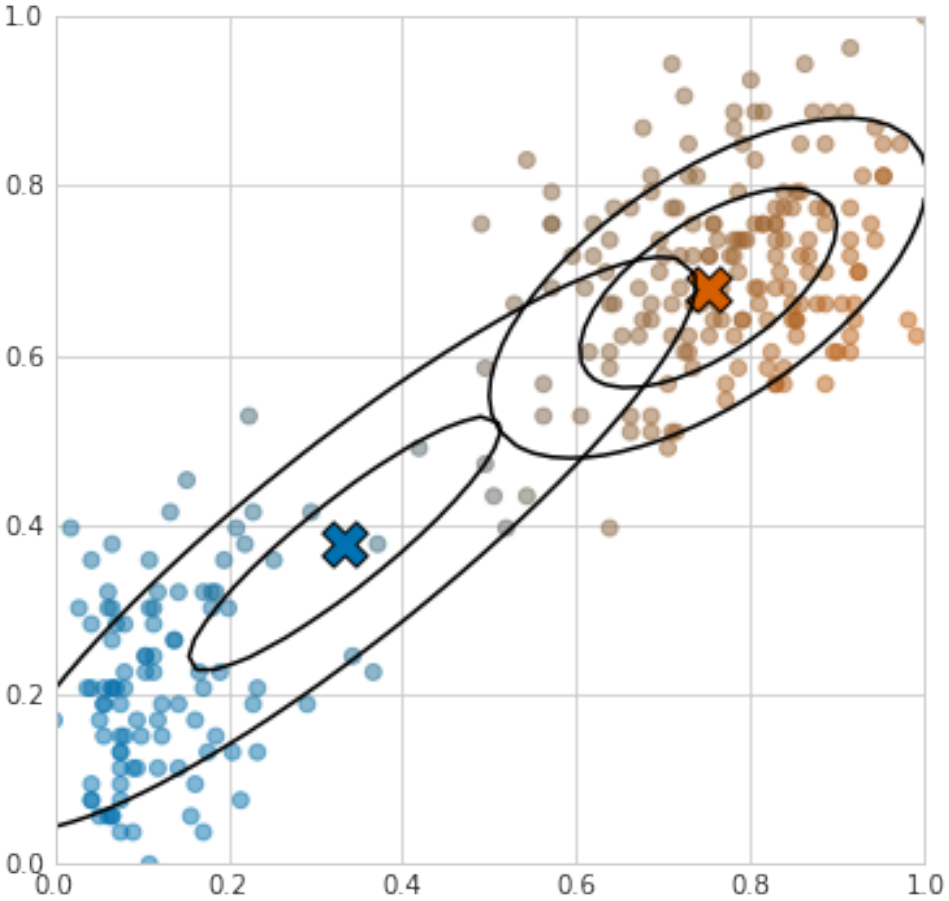

```
Iteration 8: log-likelihood = 139.88, improvement = 4.14
```



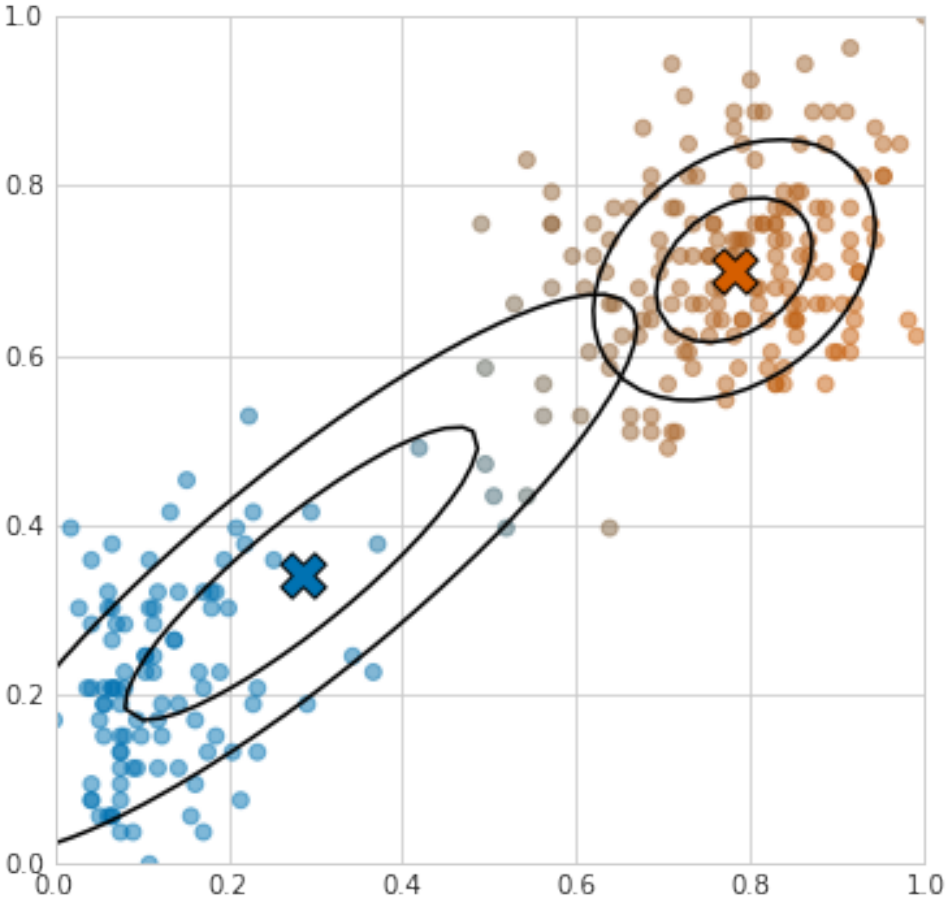

```
Iteration 9: log-likelihood = 150.67, improvement = 10.79
```



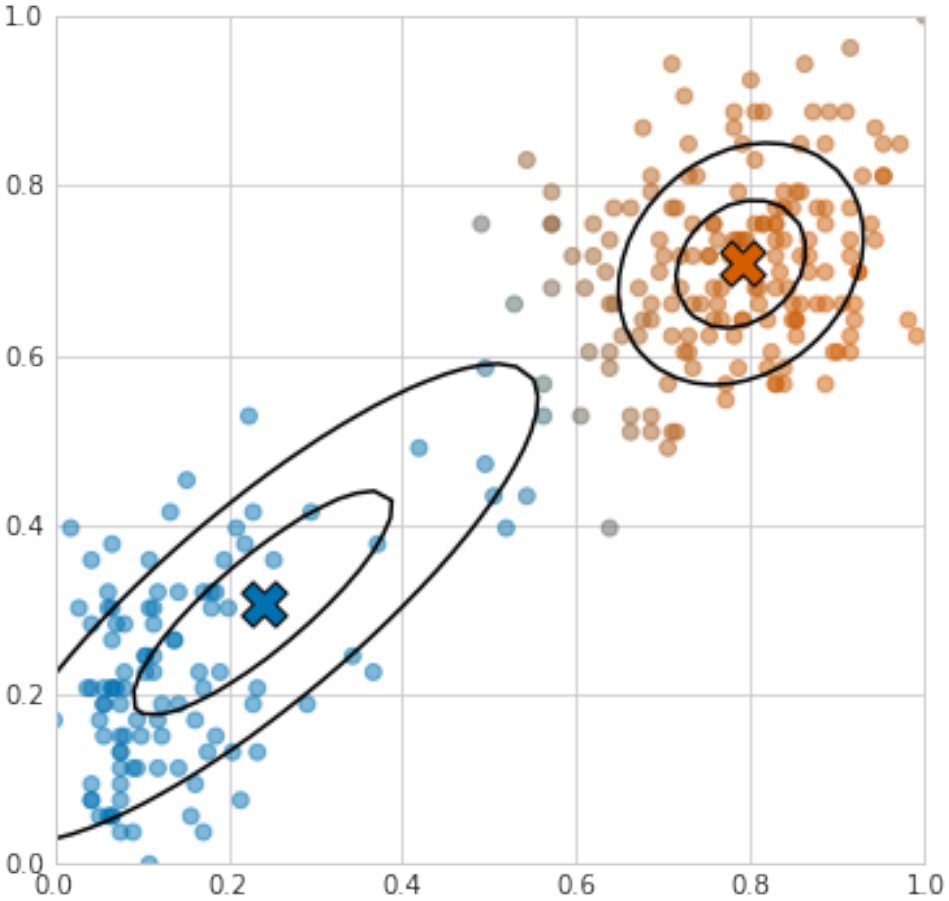

```
Iteration 10: log-likelihood = 181.12, improvement = 30.45
```



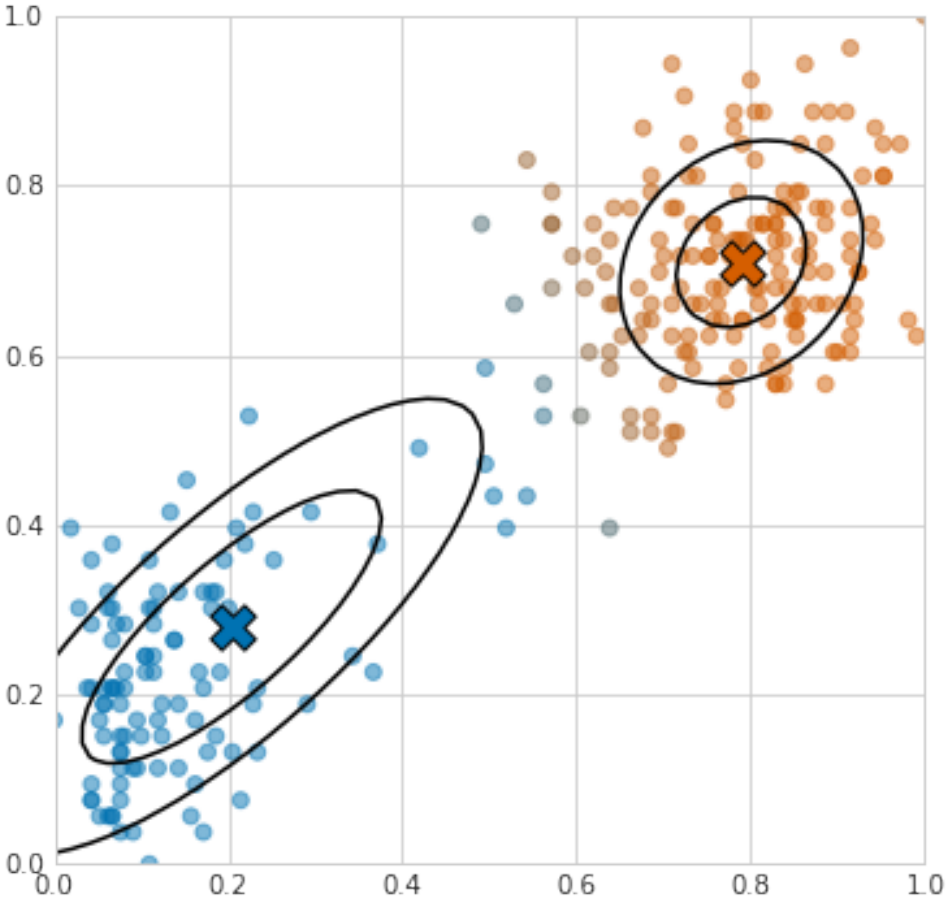

```
Iteration 11: log-likelihood = 220.93, improvement = 39.81
```



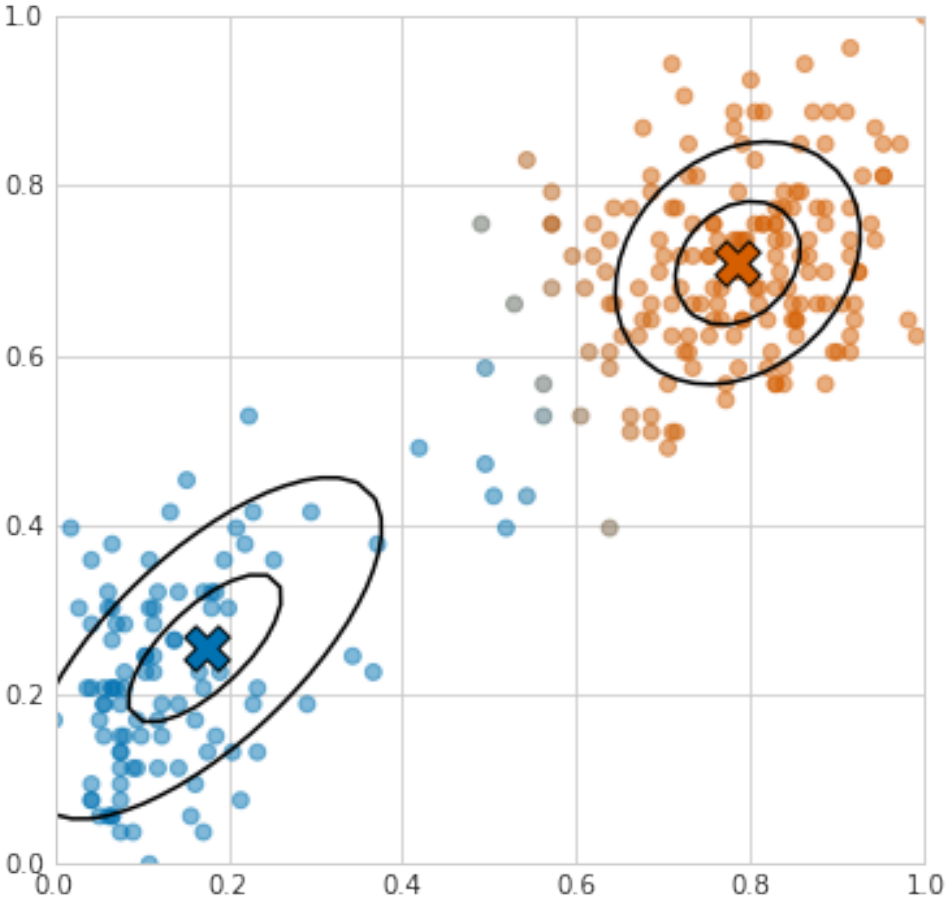

```
Iteration 12: log-likelihood = 234.06, improvement = 13.14
```



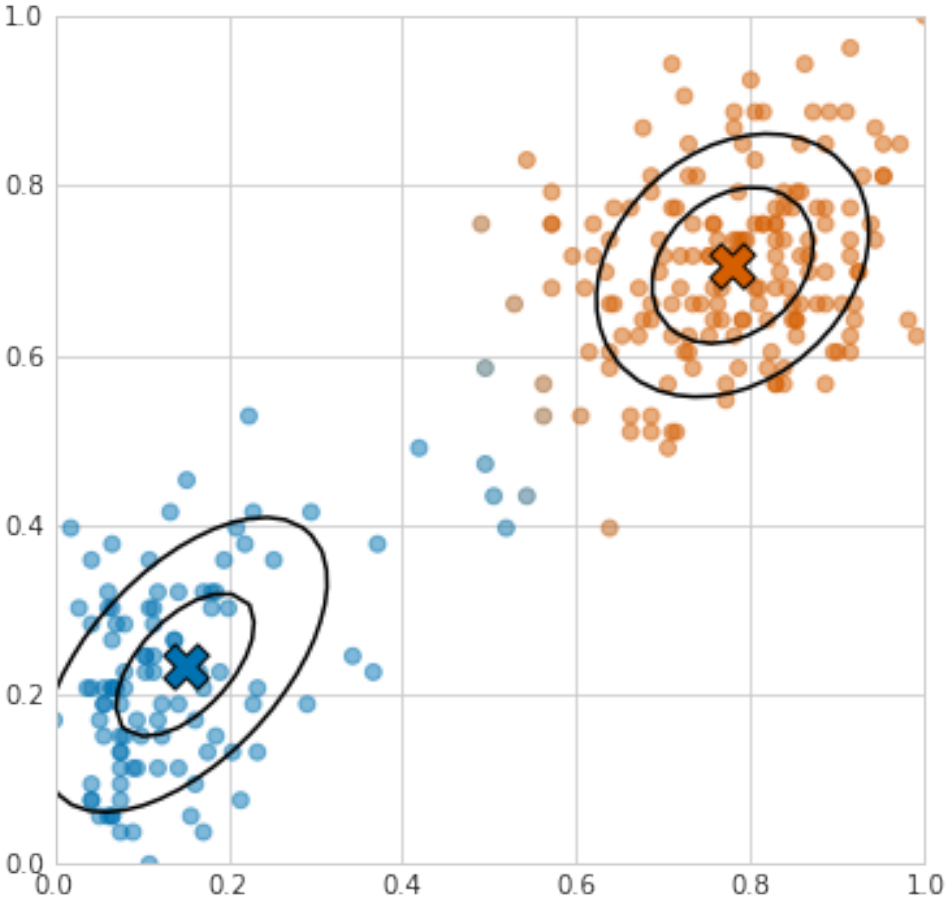

```
Iteration 13: log-likelihood = 244.83, improvement = 10.77
```



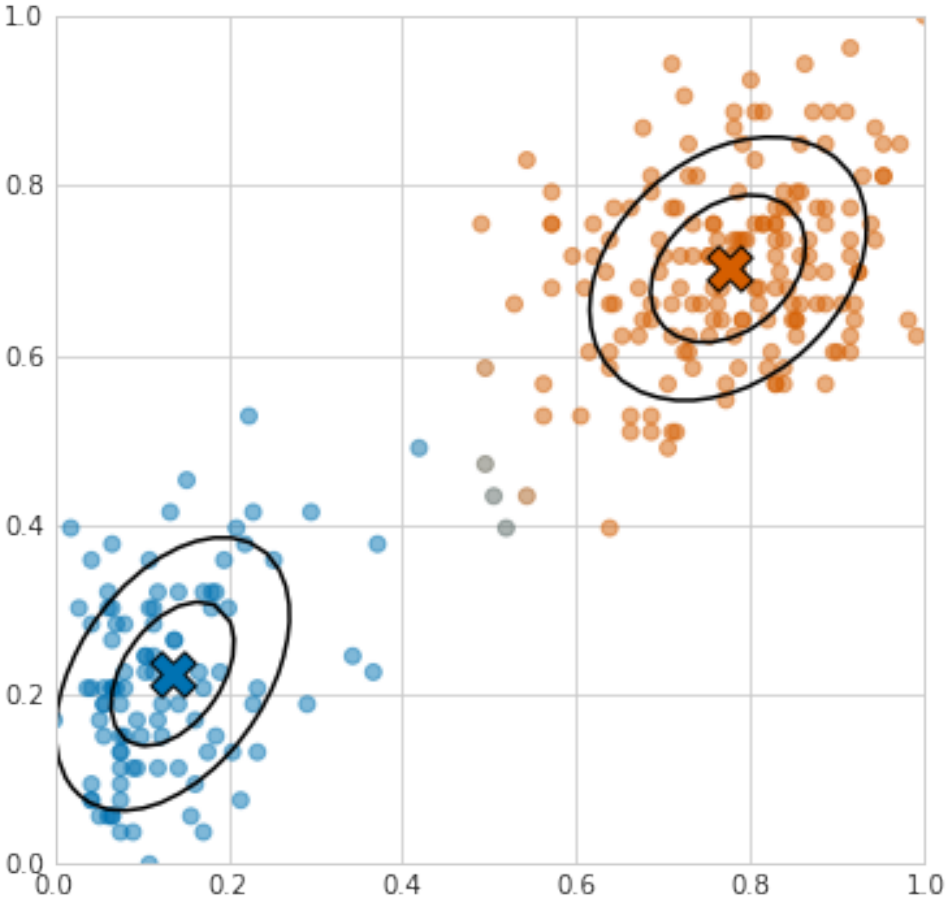

```
Iteration 14: log-likelihood = 258.67, improvement = 13.84
```



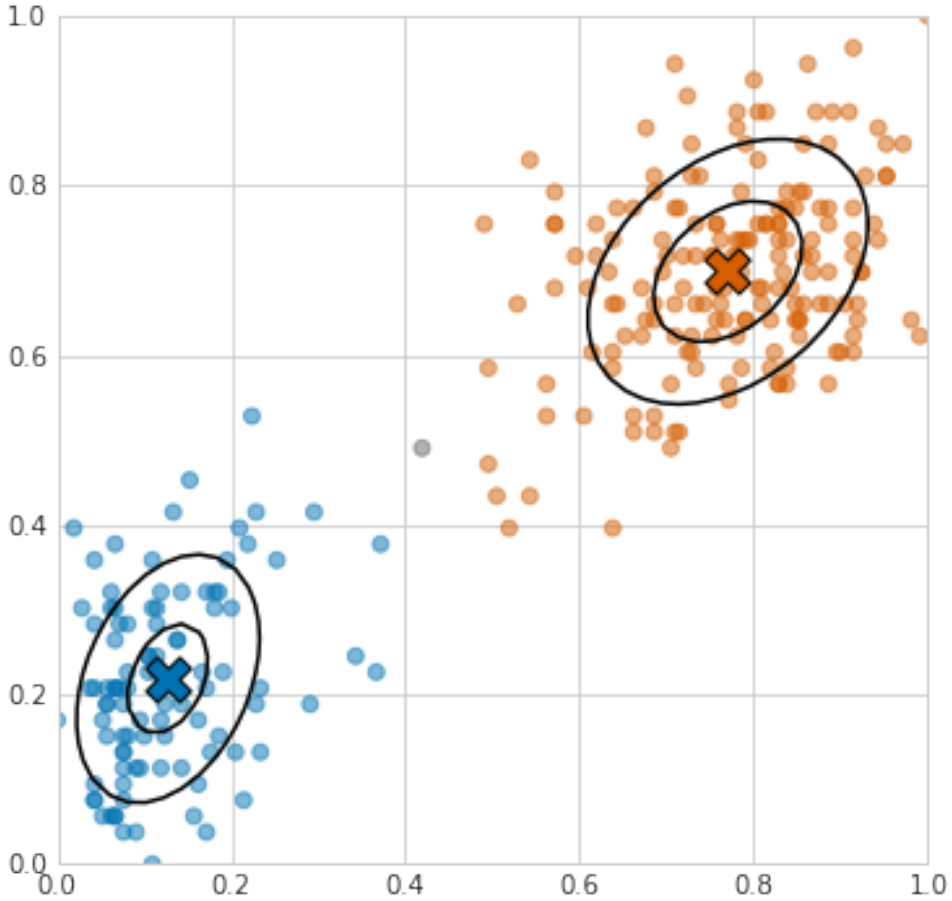

```
Iteration 15: log-likelihood = 272.91, improvement = 14.23
```



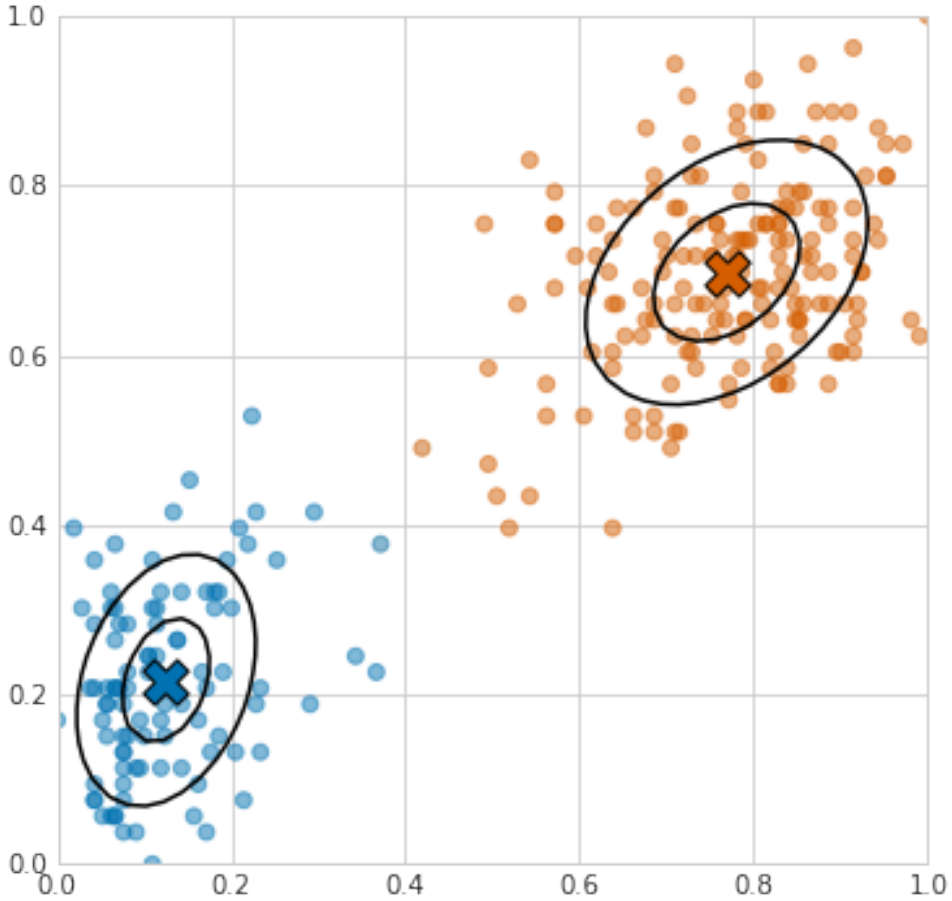

```
Iteration 16: log-likelihood = 284.29, improvement = 11.38
```



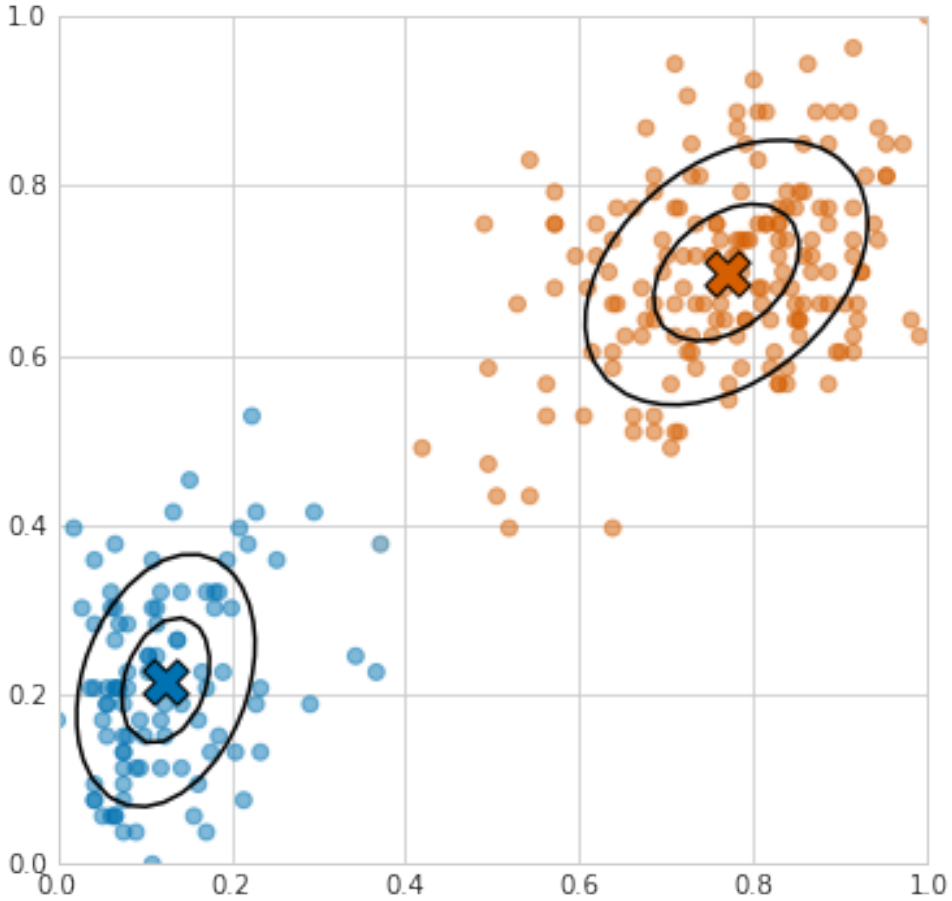

```
Iteration 17: log-likelihood = 289.94, improvement = 5.65
```



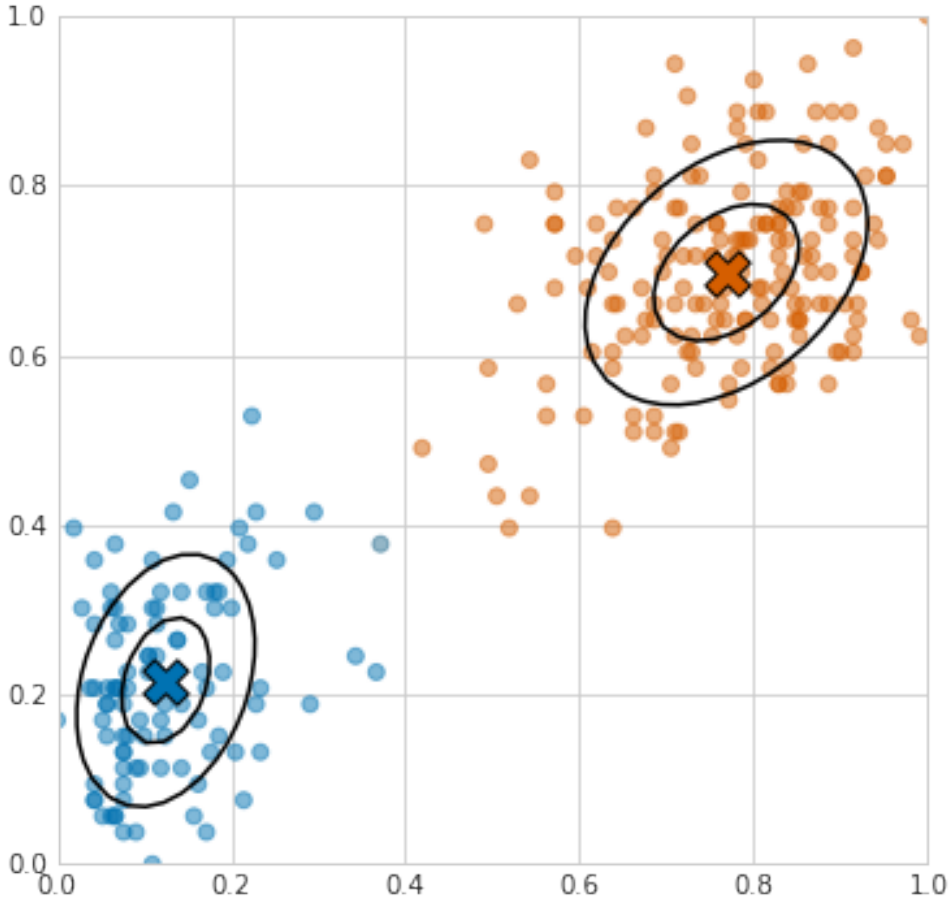

```
Iteration 18: log-likelihood = 290.39, improvement = 0.45
```




```
Iteration 19: log-likelihood = 290.41, improvement = 0.01
```




```
Iteration 20: log-likelihood = 290.41, improvement = 0.00
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Machine Learning Worksheet 12

Variational Inference

1 KL divergence

Problem 1: Compute the KL divergence between two Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ with diagonal covariance matrices.

Hint: If you use the facts you know about normal distribution, you can save yourself a lot of work before taking the straightforward path.

Problem 2: Consider that $p(\boldsymbol{x})$ is some arbitrary fixed distribution that we wish to approximate using an isotropic Gaussian distribution $q(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{I})$ (covariance matrix is identity matrix).

By writing down the KL divergence $\mathbb{KL}(p\|q)$ and then differentiating w.r.t. $\boldsymbol{\mu}$, show that the optimal setting of the parameter is

$$\boldsymbol{\mu}^* = \arg\min_{\boldsymbol{\mu}} \mathbb{KL}(p\|q) = \mathbb{E}_p[\boldsymbol{x}]$$

2 Mean-field variational inference

Consider a very simple probabilistic model with a 2-D latent variable $\boldsymbol{z} \in \mathbb{R}^2$ and an observed variable $x \in \mathbb{R}$.

The prior over the latent variable is

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{0}, \boldsymbol{I}) = \mathcal{N}(z_1 \mid 0, 1) \cdot \mathcal{N}(z_2 \mid 0, 1),$$

and the likelihood is

$$p(x \mid \boldsymbol{z}) = \mathcal{N}(x \mid \boldsymbol{\theta}^T \boldsymbol{z}, 1),$$

where $\boldsymbol{\theta} \in \mathbb{R}^2$ is a known and fixed parameter.

Problem 3: Write down the true posterior distribution $p(\boldsymbol{z} \mid x)$.

Can the posterior be factorized over z_1 and z_2? (i.e. can it be expressed as $p(z_1 \mid x)p(z_2 \mid x)$?)

Problem 4: We approximate the true posterior using a mean-field variational distribution

$$q(\boldsymbol{z}) = q_1(z_1)q_2(z_2) = \mathcal{N}(z_1 \mid m_1, s_1^2) \cdot \mathcal{N}(z_2 \mid m_2, s_2^2)$$

Your task is to derive the optimal updates for q_1 and q_2.

Is $q(\boldsymbol{z})$ able to match the true posterior $p(\boldsymbol{z} \mid x)$?

Upload a single PDF file with your solution to Moodle by 29.01.2018, 9:59am CET. The solution must be typeset (using Latex, Word, etc.). Handwritten solutions are not accepted.