Machine Learning Exam

Christoffer Thrysøe - dfv107
Exam number: 347

January 27, 2017

1. In a galaxy far, far away

Question 1.1

The file `assignment1.py` implements the subroutines `get_variance` and `MSE`, which outputs the below results
The variance σ^2_{red} of the redshifts, detailed in the training data, was calculated using the following:

$$\sigma^2_{\text{red}} = \frac{1}{N}\sum_{i=0}^{N}(X_i - \mu)^2 \tag{1}$$

where μ is the mean of the redshifts defined as:

$$\mu = \frac{1}{N}\sum_{i=0}^{N} x_i \tag{2}$$

The variance of the dataset `ML2016SpectroscopicRedshiftsTrain.dt` was measured to be:

$$\sigma^2_{\text{red}} = 0.0106043856253$$

The mean-squared-error (MSE) is defined as the following:

$$\text{MSE} = \frac{1}{N}\sum_{i=0}^{N}(\hat{y_i} - y_i)^2 \tag{3}$$

where $\hat{y}$ denotes the estimated value of the actual target value y. MSE has been applied on the SDSS predictions on the test data and the actual values. The returned MSE was the following:

$$\text{MSE} = 0.000812319455494$$

Question 1.2

The subroutine `getError` produces the below results
Linear regression was performed by applying least squares, however the results were very unsatisfying,

probably due to the linear dependency in the data. Therefore i have used the weights described in the announcement on Absalon. From these weights i have estimated the target value $\hat{y}$ using the following:

$$\hat{y_i} = wx_i + w_0 \tag{4}$$

The MSE on the training and test data is defined below:

$$\begin{aligned}\text{MSE}_{\text{train}} &= 0.0018621\\ \text{MSE}_{\text{test}} &= 0.00186407\end{aligned}$$

Normalizing the error w.r.t the variance yields the following result:

$$\frac{\text{MSE}_{\text{train}}}{\sigma^2_{\text{red}}} = \frac{0.0018621}{0.0106043856253} = 0.17559716006 \tag{5}$$

One can say that the MSE estimates the error variance. If we have that the points are distributed at random, the variance and the MSE of the linear fit will be close to identical, thus (5) will be 1. The only way we can have that (5) is greater than 1 is if the data is not linear as the otherwise the MSE will be less than the variance. If the data is perfectly linear and we apply linear regression, then the only way we can have that (5) is greater than 1 is if the linear fit is worse than the horizontal mean line. However this will not be possible when applying least squares, except if the data don't fit the model.

Question 1.3

The file `KNN.py` implements the k-nearest neighbour and cross-validation algorithms described in this section.

For the non-linear regression method i have chosen the k-nearest neighbour. I have used the same implementation as in the previous weekly assignment, however for regression instead of classification. I chose the k-nearest neighbour algorithm is that it usually makes good predictions on the data, without making any assumptions about the data. As the dataset is quite big and consists of many features, this is wanted as finding trends in data can be tough due to the dimensionality. There is although a set of shortcomings that follows the k-nearest neighbour. Although we may not need to make any assumptions on the data, we also assume that each feature contribute equally to the overall estimation, this is however rarely true. The computation time is also very high, when the dataset is big. One last thing which should be accounted for is that each data feature may measure on different scales, which results in features with high numerical ranges dominating low ones. To counter this, the training data has been normalized to exhibit zero-mean and unit-variance. This affine mapping obtained by the training data, has also been applied to the training data.

The process of obtaining the k-nearest neighbours of a sample is the same: The euclidean distance from each point in question is taken to the rest of the points, the top k points with the lowest distances are marked as neighbours.The regression variant doesn't use a voting phase on the target-values of the k-nearest neighbours, but the mean of their target-values instead, as shown in equation (6).

$$\hat{y} = \frac{1}{K}\sum_{i=0}^{K} y(x_i) \tag{6}$$

where y denotes the target value associated with the independent variable x.

For hyper parameter tuning, i.e. finding the optimal value of k, i have used a 5-fold cross-validation. The training set of 5000 samples is split into 5 equally sized partitions, each of these partitions is by turn used as a validation set and the rest as a training set. The k-nearest neighbour algorithm is applied to each of the partitions, and the average MSE error over these folds determines the average error for the given value

k. The k with the lowest average error is used as the parameter of the k-nearest neighbour algorithm.

The cross validation has been performed for the following values of k:

$$k \in [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25]$$

The average error over the five folds is illustrated in figure 1.

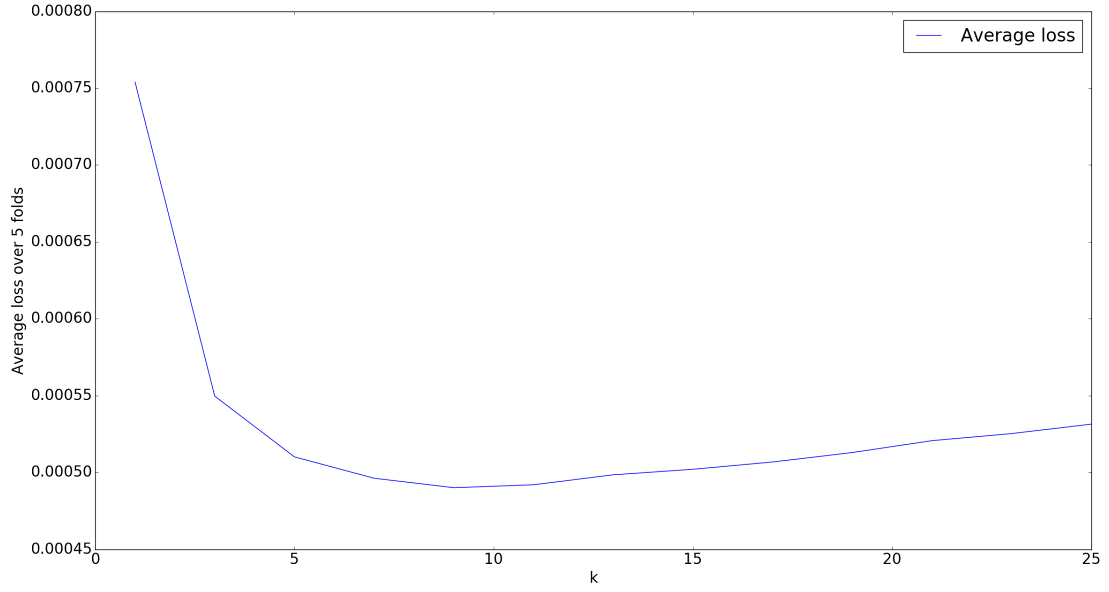

Figure 1: Average MSE error over 5 folds shown for increasing parameter k. The lowest average error is k=9

The smallest MSE averaged over 5-folds was achieved using the parameter $k = 9$, therefore 9 is the hyper parameter chosen. The following shows the result of running the 9-nearest-neighbour algorithm on the training and testing data. When applying k-nearest neighbour on the training data, the closest neighbour will always be itself, this is not the case for the test data, as the points are introduced to a dataset where they don't exists. The results are the following:

$$\text{MSE}_{\text{train}} = 0.000465310841171$$
$$\text{MSE}_{\text{test}} = 0.000588914605838$$

The above results both show an MSE, which is far lower than the liner regression model. Thus the non-linear 9-nearest-neighbour has a better performance, and provides a good generalization on the data.

2. Weed

Question 2.1

The file `logRes.py` implements the logistic regression algorithm described below.
For this assignment I have used the logistic regression algorithm implemented in previous assignments. The implementation uses the the steepest decent variant of gradient decent, where the gradient g is updated as

the following

$$g = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n x_n}{1 + e^{y_n w^T x_n}} \tag{7}$$

and the weights are updated by:

$$w = w - \epsilon g \tag{8}$$

where ϵ is the learning rate of the gradient decent. The target class is evaluated as the following:

$$h(x) = \theta(\mathbf{w}^T \mathbf{x})$$

where θ is the sigmoid function, which estimates the probability of the given class. Since we have a binary decision problem, we can define the decision boundary as the following:

$$h(x) = \begin{cases} +1 & \text{for } h(x) \geq 0.5 \\ -1 & \text{for } h(x) < 0.5 \end{cases}$$

For the model, i have used $\epsilon = 0.00001$. The small learning rate was used as values for the data was large, and a larger learning rate caused the gradient to grow large and cause overflow. The model has been trained using 5000 iterations of the steepest descend algorithm.
The zero-one-loss is reported on the training and testing data using the model trained using the training data. The zero-one-loss is defined as followed:

$$l(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{otherwise} \end{cases}$$

The errors on the training and test data are as followed:

$$l(y, \hat{y})_{train} = 26$$
$$l(y, \hat{y})_{test} = 20$$

where the training data consists of 1000 examples and the test data 574 examples. Although the above results may have a high error, the model exhibits good generalization on the test data as the empirical loss (average loss) is somewhat close to the expected loss.

Question 2.2

The file `svmc.py` implements the below experiments, grid search and cross-validation.
We wish to determine G, which is the following quantity:

$$G = \left\{ \min_{x_j, y_j) \in S \wedge y_i \neq y_j} \{||x_i - x_j||\} \,|(x_i, y_i) \in S \right\} \tag{9}$$

G is calculated by iterating over the training data, and for each point creating a list of training data points sorted by closest distance. The sorted list is then iterated, when a training point where $y_i \neq y_j$ is encountered, the distance $||x_i - x_j||$ is accumulated. The estimate σ_{Jaakkola} is given by:

$$\sigma_{\text{Jaakkola}} = \text{median}(G) \tag{10}$$

The bandwidth parameter γ can be derived by the following:

$$\sigma_{\text{Jaakkola}} = \sqrt{1/(2\gamma_{\text{Jaakkola}})} \Rightarrow$$
$$\gamma_{\text{Jaakkola}} = 1/2\sigma^2_{\text{Jaakkola}}$$

I estimated $\sigma_{\text{Jaakkola}} = 609.035300487$ therefore γ_{Jaakkola} is estimated by:

$$\gamma_{\text{Jaakkola}} = 1/2 \times 609.035300487^2 = 1.3479 \times 10^{-06} \tag{11}$$

For the task, the library LIBSVM for python was used for training and classification. The library implements the desired C-SVM. The C parameter determines the softness of the margin, where a small C gives a soft margin and a large C gives a hard margin. Thus a large C will try to classify as much data as possible correctly, yielding a smaller margin, where a soft margin will try to maximize the margin allowing misclassified variables. For the SVM the radial gaussian kernel is applied, which is defined as followed:

$$k(x, z) = exp(-\gamma||x - z||^2) \tag{12}$$

γ is the width of the gaussian kernel, therefore a a small γ results in large variance. For parameter tuning, grid search was applied, where each parameter setting was evaluated by performing 5-fold cross-validation. The choosing of parameters which yielded the lowest average zero - one loss over the 5-folds was inserted into the grid. The grid index with the lowest average zero - one loss, was determined as parameter settings. The values for C and γ were picked from the following set:

$$(C, \gamma) \in \left\{b^{-3}, b^{-2}, b^{-1}, 1, b, b^2, b^3, b^4, b^5, b^6\right\} \times \left\{\gamma_{Jaakkola} \cdot b^i | i \in \{-3, -2, -1, 0, 1, 2, 3\}\right\}$$

where $b \in \{2, e, 10\}$. Running the 5-fold cross validation the parameters for C and γ which gave the lowest average zero - one loss was:

$$C = b^4$$
$$\gamma = \gamma_{Jaakkola} \cdot b^{-1}$$

The lowest error on the training data was measured when b was 2 or e. The above parameters was obtained during training, below is the error of applying the SVM, with the above parameters on the training and test data.

$$L(y, \hat{y})_{\text{train}} = 0.015$$
$$L(y, \hat{y})_{\text{test}} = 0.0348$$

It is notable that the training error is very low compared to error achieved by the logistic regression algorithm. However the test error is very high compared to the training error, which means that the model does not generalize well on the unseen data, as the expected loss does not track the empirical loss.

Question 2.3

The normalization of both procedures is defined respectively in the files they were implemented in.
To achieve zero-mean and unit-variance, we apply a linear mapping from the input data to the normalized output data, that is of the form $R^d \rightarrow R^d$. Thus we wish to determine some normalization function $f_{a,b}$:

$$X_{f_{a,b}} = \{ax_i + b | x_i \in X\} \tag{13}$$

such that $\mu(X_{f_{a,b}}) = 0$ and $Var(X_{f_{a,b}}) = 1$
To achieve this, the mean of each input feature is found and subtracted. The mean is calculated by:

$$\mu_{\mathbf{X}} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Next the variance is found of each feature. The variance is calculated by:

$$Var(X) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu_{\mathbf{X}})^2$$

The standard deviation σ is equal to the square root of the variance and is used to normalize the data. The normalization can then be expressed as the following:

$$X_{f_{a,b}} = \frac{1}{\sigma_{\mathbf{X}}}\mathbf{x} + \frac{-\mu_{\mathbf{X}}}{\sigma_{\mathbf{X}}} = \frac{\mathbf{x} - \mu_{\mathbf{x}}}{\sigma_{\mathbf{X}}} \tag{14}$$

For the below algorithms i will first state how i predict the normalization will effect the estimation, then results of the algorithms on the normalized data.

Logistic regression

First i will determine how the estimator is affected by the normalization. Given a logistic regression applied to normalized data x' we include the affine transformation in the calculation:

$$\begin{aligned} w^T x' &= w^T \cdot (ax_i + b) \\ &= w'^T x \end{aligned}$$

when $x_0 = w_0 + \sum_{i=0}^{d} b_i w_i$ and $w'_i = aw_i$. Therefore the weights obtained using normalized data, can be obtained by using the affine transformation parameters on the weights obtained from performing logistic regression without normalization. Thus the normalization shouldn't effect the classification procedure.
Logistic regression uses gradient decent and this procedure however effects from data normalization. The normalization can be said to normalize the gradient surface. Normalization is often used in the context of optimization problems, where computations of large gradients are expensive and may cause overflow. The normalization also makes convergence faster and ensures that weights are updated somewhat equally. Given the above reasoning for normalizatio,n before applying logistic regression, I predict that the result will be somewhat equal to using the non-normalized data. Since the gradients have been normalized, I will have to adjust the learning rate ϵ and number of iterations.
Applying normalization to the data before performing logistic regression worsens the generalization in contrast to the SMV. Below is the zero - one loss when normalization is applied to the dataset:

$$\begin{aligned} l(y, \hat{y})_{train} &= 57 \\ l(y, \hat{y})_{test} &= 44 \end{aligned}$$

As evident from the above errors, the logistic regression estimator performs worse when applied to normalized data. This was not expected given the above discussion.

Support vector machines

Achieving zero-mean and unit-variance when using support vector machines is useful in multiple ways. Kernels often depend on distances between points e.g the Gaussian kernel. To avoid larger numeric ranges dominating small ones, it's necessary to scale the features to the same range. Like with logistic regression, SVM's solve an optimization problem, which benefits from scaled features. Due to the feature scaling, i predict that the error will be lower, when using normalized data.

Applying zero mean and unit variance to the dataset before running the training the support vector machine leads to a different loss and different choice of hyper parameters. $b = 2$ is still the best setting for b. The found parameters using grid search are as followed:

$$\begin{aligned} C &= b^5 \\ \gamma &= \gamma_{Jaakkola} \cdot b^{-3} \end{aligned}$$

The linear mapping was achieved on the training set, the same mapping is used on test set. The training and test error is as followed:

$$\begin{aligned} L(y, \hat{y})_{\text{trainnorm}} &= 0.015 \\ L(y, \hat{y})_{\text{testnorm}} &= 0.0278 \end{aligned}$$

We note from the above errors that the normalized data yields the same error on the training set, however as expected, we achieve a much better generalization of the data, as the test error is lower.

Random Forest

Intuitively there is no need for normalization as the recursive splitting is not determined by the magnitude of the features, but by an impurity measure. The threshold at each node is feature scale invariant, and a scaling of the features will only reflect in a shift of this threshold, thus not impacting the split.
For random forest we split the data, maximizing $G_{d,\theta}(S)$ the information gain. If we consider a split, on a data point x normalized using the affine transformation $f_{a,b}(x) = x'$. The split is determined by $(d, \theta) = (d, x'_d)$, the following is achieved:

$$\theta = x'_d = f_{a,b}(x)_d = ax_d + b_d \Leftrightarrow x_d = (\theta - b_j)/a \tag{15}$$

Thus the parameters of the affine transformation, uniquely defines the threshold, and therefore have no influence on the classification.

Question 2.4

The file `pca.py` implements the principle component analysis described below
The implementation of the PCA algorithm is located in the file `pca.py`. The algorithm performs the following steps:

1. Separate features and target values of each data set
2. For each feature, calculate the mean and withdraw it from the data points
3. Compute the covariance matrix of the mean-less data
4. Compute eigenvalues and eigenvectors of the covariance matrix
5. Sort the eigenvectors based on the corresponding eigenvalue from largest to smallest
6. Take the top k (dimension to reduce to) eigenvectors. These correspond to the k principle components
7. Project the data into k dimensions by multiplying the principle components and the mean-less data

Eigenspectrum

To compute the eigenspectrum the eigenvalues have been listed in descending order, and plotted on a logarithmic scale as shown in figure 2.

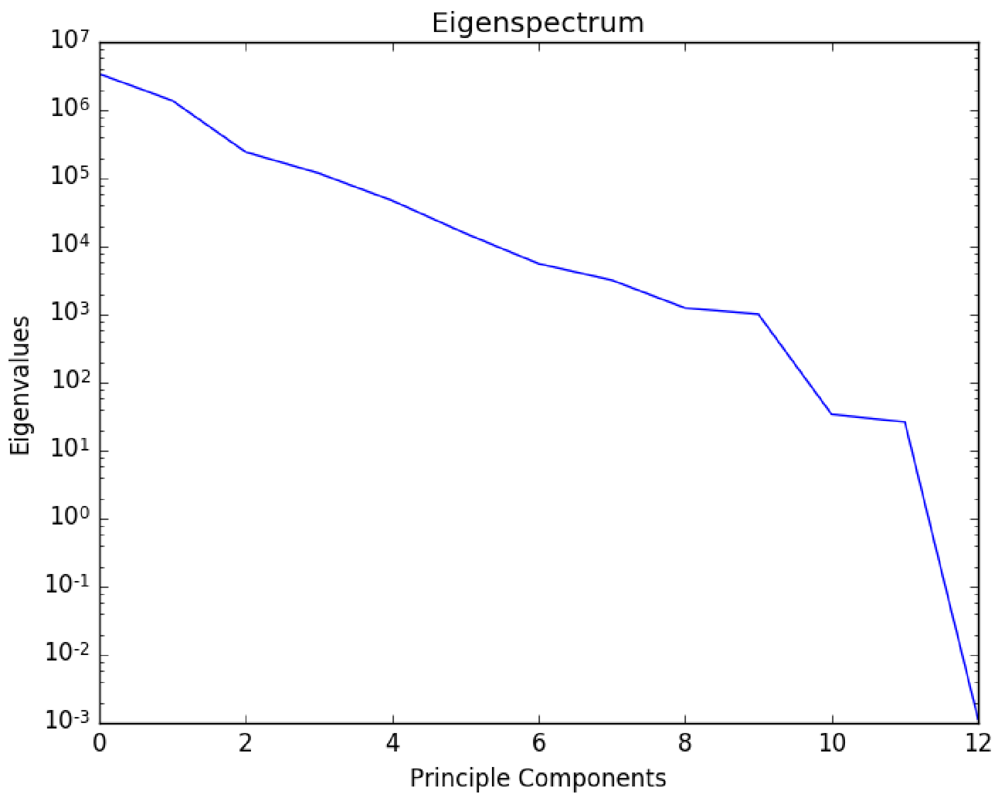

Figure 2: Variance of each principle component in descending order. 90% of the variance is described after the second principle component.

The number of principle components necessary to explain 90 % of the variance was determined to be **2 components**.
From figure 2 it is evident that the variance drops linearly until about the 9th component where the variance plummets.

Projection onto two dimensions

The data has been projected down to two dimensions, by performing PCA with k = 2, that is multiplying the first two principle components onto the mean-less data. Figure 3 shows a scatter plot for the for crops and weed.

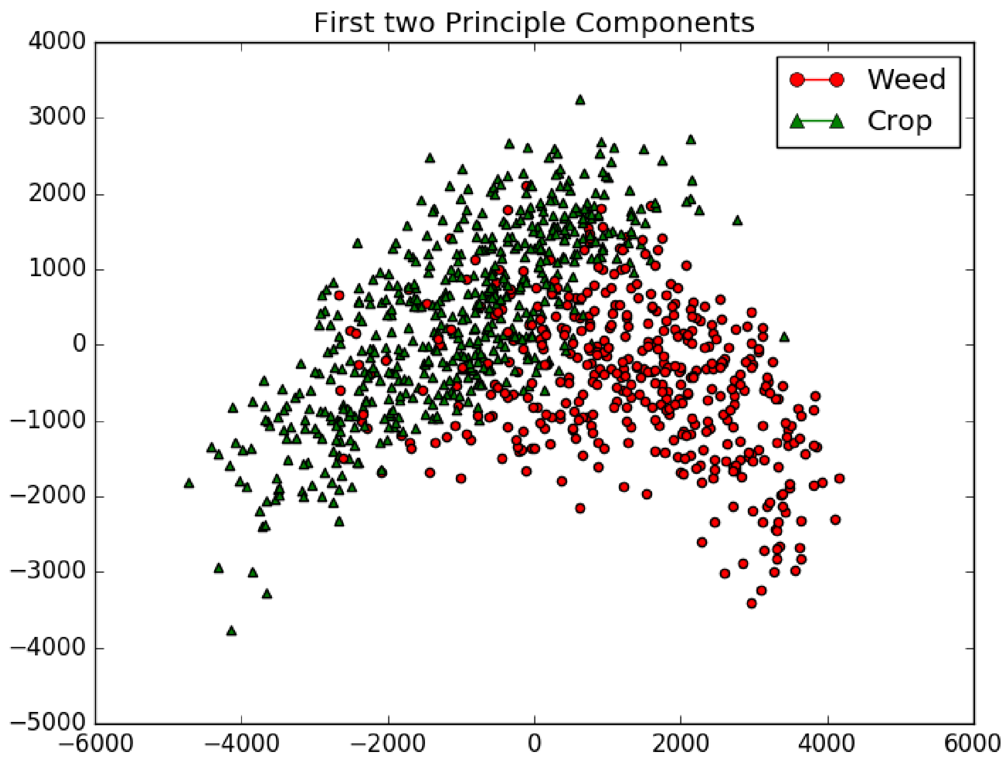

Figure 3: Scatter plot of the data projected onto the two first principal components.

The below table shows the correlation between the classification and the colour which they were plotted in. :

Class	Colour
Weed	*Red*
Crop	*Green*

From figure 3 we can see that the projection onto the two first principal components, describes the data well, as the data is somewhat divided into two clusters. Many of the points are well separated, however some red points (weed) lie in the green (crop) cluster, which could cause misclassification of crop weed images as weed-less crop images.

Question 2.5

The implementation of k-means clustering is included in the file `kmean.py`.
The algorithm works as followed:

1. k cluster points are initialized, with a given position in the data. For this assignment the data is initialized to have the same position as the first two data points.
2. Each data point is assigned the cluster with the smallest euclidean distance.
3. For each cluster, the mean of the assigned points are calculated and the cluster position is then updated with the mean.

4. The two steps above are repeated until the clusters no longer move. The positions of these clusters are then returned.

Figure 4 shows the cluster position projected onto the two first principle components, using the same projection calculation as assignment 3.2. The starting positions of the two centroids correspond to the first two data points. All clusters converged after 30 iterations.

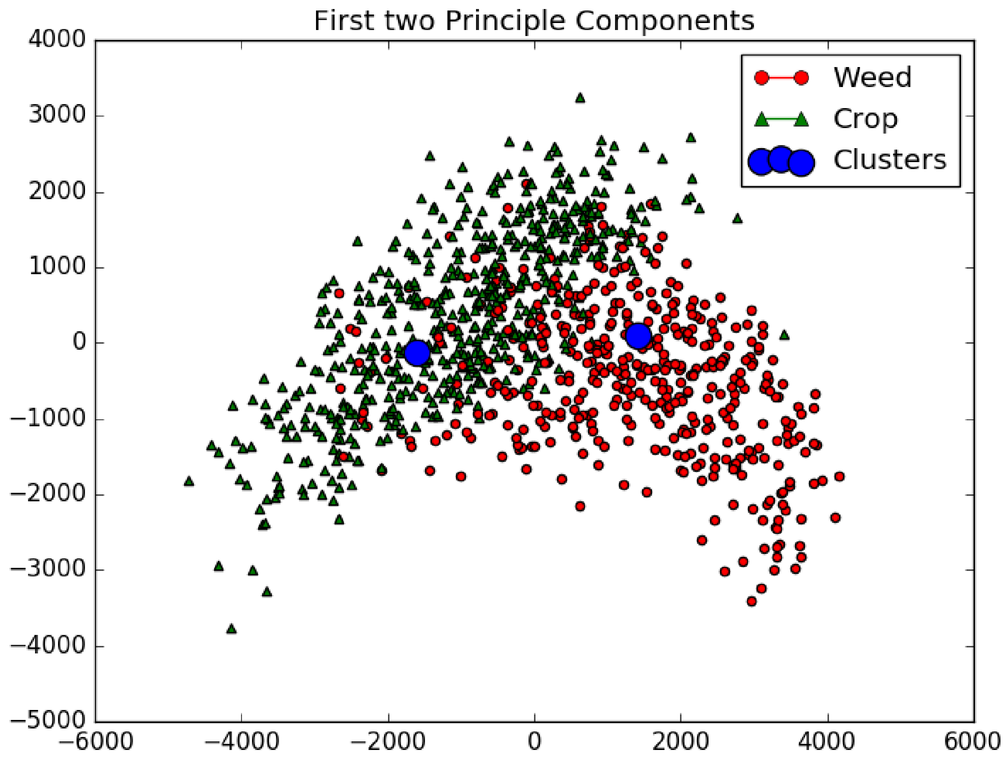

Figure 4: The result of running 4-clustering on the crop training dataset, where the position of the clusters (blue dots) have been projected onto the first two principle components.

I also tried to perform the clustering with random data points as initialization of the centroids instead of the first two. However every initialization seemed to result in the centroids being located in the same spot, although with different number of iterations before converging. I believe i got meaningful clusters, as the centroids lie in each cluster, however it's clear that the overlapping data points could cause a high degree of misclassification. The clusters of points lie very close to each other. A clearer division of the clusters would lead to better placements of the centroids.

3. Generalization bound for learning with multiple feature mappings

Question 1

First we note that the VC-dimension of a hyperplane in dimension d is $d+1$ (Abu-Mostafa et al., 2012 page 52), that is we can shatter at most $d+1$ points. When performing a mapping from an x space to a z feature space, we map a d dimensional vector, representing our input to a new $\tilde{d}$ feature space:

$$X = (x_0, x_1, ..., x_d) \xrightarrow{\Phi} z = (z_0, z_1, ..., z_{\tilde{d}})$$

The Q-th order polynomial transformation $\Phi_Q(x)$ is used, listing all monomials of degree zero up to Q. The dimension of $\tilde{d}$ is given by the number of monomials from the Q-th order polynomial transformation. This number is defined as:

$$\tilde{d} = \binom{d+Q}{Q} - 1 \tag{16}$$

The minus one is because we don't count 1 as a dimension. We now have a $\tilde{d}$ dimensional feature space. The total number that the hyperplane can shatter in this feature space and the VC-dimension is therefore:

$$d_{vc} \leq \tilde{d} + 1 \tag{17}$$

We now have the inequality for the wanted bound:

$$\left(\binom{d+Q}{Q} - 1\right) + 1 \leq (Q+1)d^Q \tag{18}$$

which holds for all $Q, d \geq 0$

Question 2

From theorem 3.7 in the lecture notes we have the following VC generalization bound.

$$Pr\left[\exists h \in \mathcal{H} : L(h) \geq L(\hat{h}, S) + \sqrt{\frac{8\ \ln(2((2n)^{d_{vc}} + 1)/\delta}{n}}\right] \leq \delta \tag{19}$$

which we can write the complement of for all $\mathcal{H}$ to arrive at the following high probability bound on $L(h)$.

$$Pr\left[\forall h \in \mathcal{H} : L(h) \leq L(\hat{h}, S) + \sqrt{\frac{8\ \ln(2((2n)^{d_{vc}} + 1)/\delta}{n}}\right] \geq 1 - \delta \tag{20}$$

which we can write in terms of the bounded vc dimension, where $d_{vc} \leq (Q+t)d^Q$

$$Pr\left[\forall h \in \mathcal{H} : L(h) \leq L(\hat{h}, S) + \sqrt{\frac{8\ \ln(2((2n)^{(Q+t)d^Q} + 1)/\delta}{n}}\right] \geq 1 - \delta \tag{21}$$

Question 3

The norm for an n-dimensional vector is defined as followed (Lecture slides on kernels):

$$||x|| = \sqrt{x_1^2 + ... + x_n^2}$$

We can bound $||\Phi_Q(x)||$ by $||\Phi_Q^+(x)||$ as $\Phi_Q^+(x)$ will contain duplicate monomials of $\Phi_Q(x)$ and as each individual monomial norm will be positive we can write $||\Phi_Q(x)|| \leq ||\Phi_Q^+(x)||$. We note that we can write the norm of a vector in terms of its inner product:

$$||x|| = \sqrt{\langle x, x\rangle} \tag{22}$$

Therefore we can write the following for the norm of $\Phi_Q^+(x)$:

$$||\Phi_Q^+(x)|| = \sqrt{\langle \Phi_Q^+(x), \Phi_Q^+(x)\rangle} \tag{23}$$

Using the kernel trick we can write the above using the x space:

$$||\Phi_Q^+(x)|| = \sqrt{1 + \sum_{i=1}^{Q} (x^T x)^i} \tag{24}$$

Thus we can conclude the following bound on $||\Phi_Q(x)||$:

$$||\Phi_Q(x)|| \leq \sqrt{1 + \sum_{i=1}^{Q} (x^T x)^i} \tag{25}$$

For each monomial at some order we have the the norm of the monomial cannot exceed 1 as the monomial is some multiplication combination of x and since $||x|| \leq 1$ we can say the same for the rest of the monomials. Thus if the number of monomials is $|\Phi_Q|$ then the norm is bounded as:

$$||\Phi_Q|| \leq \sqrt{|\Phi_Q|} \tag{26}$$

Question 4

From theorem 3.8 in the lecture notes we have the following bound on the VC dimension for hypothesis with the width of the margin being at least ρ:

$$d_{vc}(\mathcal{H}_\rho) = \lceil R^2/\rho^2 \rceil \tag{27}$$

Where R is the maximal norm. If we set $R \leq ||\Phi_Q(x)||$ which was derived in question 3, we get the following:

$$d_{vc}(\mathcal{H}_\rho) \leq \lceil \sqrt{|\Phi_Q|}^2 / \rho^2 \rceil \tag{28}$$

following the definition from the proof of the lecture notes, we set $|\Phi_Q|/\rho^2 = i$, which we can use to express w: $\lceil ||w^2|| \rceil \cdot |\Phi_Q|$. We can therefore express the following:

$$Pr\left[\exists h \in \mathcal{H}_i : L(h) \geq L(\hat{h}, S) + \sqrt{\frac{8 \ln(2((2n)^{\lceil ||w^2|| \rceil \cdot |\Phi_Q|} + 1)/\delta_i}{n}}\right] \leq \delta_i \tag{29}$$

the same values of δ can be used here so we get the same derivation as the proof of theorem 3.9 we can therefore achieve the following generalization bound.

$$Pr\left[\exists h \in \mathcal{H} : L(h) \geq L(\hat{h}, S) + \sqrt{\frac{8 \ln(2((2n)^{\lceil ||w^2|| \rceil \cdot |\Phi_Q|} + 1)(1 + \lceil ||w^2|| \rceil \cdot |\Phi_Q|) \lceil ||w^2|| \rceil \cdot |\Phi_Q| / \delta}{n}}\right] \leq \delta \tag{30}$$

Question 5

Given the infinite hypothesis set $\mathcal{H} = \cup_{Q=1}^{\infty} \mathcal{H}_Q$ we note from assignment 1 that it is possible to shatter $d+1$ points when the feature space has dimension d. Our hypothesis set now consist of all monomials up to infinity and the feature space will therefore have dimension R^∞. Therefore by definition we can shatter all N points (represent all 2^N dichotomies), thus the VC dimension is:

$$d_{vc}(\mathcal{H}) = \infty$$

Question 6

From the bound obtained in question 5, we can re-write it for all $w \in \mathcal{H}$:

$$Pr\left[\forall h \in \mathcal{H} : L(h) \leq L(\hat{h}, S) + \sqrt{\frac{8 \ln(2((2n)^{\lceil ||w^2|| \rceil \cdot |\Phi_Q|} + 1)(1 + \lceil ||w^2|| \rceil \cdot |\Phi_Q|) \lceil ||w^2|| \rceil \cdot |\Phi_Q| / \delta}{n}}\right] \geq 1 - \delta \tag{31}$$

Question 7

Not solved.

Question 8

From question 6, we note that even if the margin is large, the generalization will be bad if the hypothesis has a large VC dimension, as we multiply with the number of monomials. The intuitive notion behind this is that we can't map our input space to an infinite feature space and still have the property that maximizing the margin will provide a good generalization. Thus it follows from corollary 3.11 that when we have an infinite VC dimension, it is impossible to achieve a high probability bound, as n goes to infinity.

Question 9

Minimizing the bound leads to a greater generalization of the model, however the flexibility of the model is limited, resulting in a bigger error. Maximizing the margin (minimizing $||w||$) leads to a better generalization of the data, as it ensures that unseen data are more likely to be classified correctly. There is a trade-off when minimizing the generalization bound, as minimizing means to limit the size of the feature space of the given hypothesis. While achieving a good generalization, the limited model may fail to capture important parts of the data which a more complex hypothesis could have, thus resulting in a bad model.