Machine Learning Final Exam

Department of Computer Science, University of Copenhagen

Dhruv Chauhan

January 25, 2017

1 In a galaxy far, far away

1.1 Data preparation

The variance of the red-shifts in the spectroscopic training data was calculated to be:

$$0.0106$$

(where from now on, unless specified, values are shown to 3 significant figures).

The MSE on the test SDSS predictions was calculated to be:

$$0.000812$$

This shows that the predictions were quite accurate.

1.2 Linear regression

The linear regression was done in Python, using the `sklearn` linear regression package. This performs an ordinary least squares linear regression. The error function is a Mean Squared Error.

The parameters of the model were (taken from the announcement):

[0.0185134,0.0479647,-0.0210943,-0.0274002,
-0.0226798,0.0064449,0.0151842,0.0120738,
0.0103486,0.00599684,-0.0294513,0.069059,
0.00630583,-0.00472042,-0.00873932,0.00311043,
0.0017252,0.00435176]

with bias term:

$$-0.801881$$

The error on the training data was calculated to be 0.00187, and on the test data was 0.00187 also. The errors normalised by the variance, σ^2_{red} were equal to 0.176 for both the test and the training data.

This normalised error gives a way of fairly comparing the different training and testing values. We use the training variance as typically we would not be able to calculate the variance of the testing data. The error falling below one signifies that the MSE result is more accurate. A lower value shows a better result, with less normalised error.

1.3 Non-Linear regression

For the non-linear regression, I chose to apply the K-nearest neighbours (KNN) algorithm. I chose this method for its simplicity (following Occam's razor), and therefore its intuitive understanding. The simplicity of the algorithm is also reflected in the single hyperparameter, k (if you consider a fixed distance metric), which means that there is less computation in tuning the hyperparameter.

I utilised the `neighbours` library from the `sklearn` package.

The KNN algorithm uses a distance metric to calculate the distance between a (set of) training point(s) and the other points. I used the Euclidian distance, given by $||\mathbf{x} - \mathbf{x}'||$, or $\sqrt{\mathbf{x}^T\mathbf{x}'}$. The algorithm works by calculating the distances from a test point to the other points, and then finding the nearest K points to that point. In a regression task, the test point is assigned the value of the mean of the nearest K neighbours.

My method involved using model selection methods such as cross-validation and grid search. Since this is model selection, cross validation was needed as I only use the training data during model selection. This gave us a better way to prevent overfitting of the training data. I used the `GridSearchCV` package from `sklearn`. The range of possible k values was given as the odd numbers between 1 and 29. This performed a 5-fold cross validation on each of the possible values of k, averaging out the resulting error (i.e. splitting the data into 5 equal chunks, using 4 as the training set, and 1 as the validation set, and then cycling through all possible 5 validation sets). The error function used in the algorithm was the mean squared error.

This method resulted in the optimum hyperparameter as $k = 7$, with a MSE of 0.00118 on the test data, and a MSE of 0.000870 on the training data.

Clearly, the KNN Regressor worked better on the training data, which is to be expected due to the model's simplicity in using the k training points' average to return the regression results - therefore the training points are bound to have a low MSE. In comparison, the training data in the linear regressor performed the same as the test data. The difference in this would be due to the nature of a linear regressor, which would 'average out' the regression line over the points, thus leading to a small MSE on both the training and test data. On the test data, the KNN Regressor did perform a bit better than the linear regressor - perhaps due to the non-linear nature of KNN capturing the underlying nature of the data slightly more accurately. Overall, I believe the KNN method worked well as I had a relatively low variance on the data - KNN can be skewed by big outliers in the data. The cross-validation helped to prevent overfitting on the data, which may have also caused the error on the test data to drop in comparison to the linear regressor.

2 Weed

2.1 Logistic Regression

The logistic regression was done in Python, using the `LogisticRegression` package from `sklearn`. This implementation uses the logistic function. I am trying to classify testing points according to binary labels (0 - weeds, 1 - crops). The algorithm tries to minimise the following function:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log e^{(-y_i(X_i^t\ w+c))+1)}$$

It also uses a coordinate descent algorithm, which is a derivative-free optimization algorithm that performs a line search in one coordinate direction for the current point in each iteration. [1]

The parameters produced by the model were:

[-0.0391283 , 0.01549887, 0.00295803, 0.00033714,
-0.00039954, 0.00348062, -0.00715483, 0.00473467,
-0.02685346, -0.05592747, -0.04317431, 0.00579407,
-0.00998736]

with bias term:

$$-1.47771341e - 05$$

The zero-one loss on the training data was: 0.0200, and on the test data: 0.0348.

2.2 Binary classification using support vector machines

I used the `sklearn` Python package, specfically the `svm.SVC` and `GridSearchCV` packages - in addition to `numpy`. Their model selection allows cross validation and grid-searching across values. It does this by splitting the training data into the training and validation sets across 5 folds and then running with the specified grid combination, for each option, finally returning the option with the lowest classification error. The error was calculated as the zero-one loss.

I used a kernel of the form:

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

Using the formula given, I calculated:

$$\sigma_{\text{Jaakkola}} = 609$$
$$\gamma_{\text{Jaakkola}} = 1.35e - 06$$

From that, the grid search looked over values of C and γ as given in the instructions, with $b = 10$.

From the grid search, the optimum hyperparameters were found to be:

$$C = 1000$$
$$\gamma = 1.35e - 8$$

The accuracy for the training and test sets is shown below:

$$\text{accuracy}_{\text{training}} = 0.978$$
$$\text{accuracy}_{\text{test}} = 0.969$$

From the accuracy results above, on both the training and test sets, the SVMs performed very well overall. This demonstrates the effectiveness of SVMs, especially in comparison to logistic regression.

2.3 Normalisation

The normalisation was performed by calculating the mean and the standard deviation of the training data. A function, f_{norm} was formed as below that would result in the training data having mean = 0 and variance = 1.

$$f_{\text{norm}} = \frac{\mathbf{x} - \mu}{\sigma}$$

This function was used to transform the training data to have the above mean and variance. The function was then also used to encode the test data, however with the *training* mean and variance (as the test mean and variance is unknown in most circumstances).

This normalised data was then used on the SVMs, resulting in:

$$\sigma_{\text{Jaakkola}} = 1.38$$
$$\gamma_{\text{Jaakkola}} = 0.0300$$

with optimum hyperparameters:

$$C = 100$$
$$\gamma = 0.0261$$

and test and training accuracy:

$$\text{accuracy}_{\text{training}} = 0.983$$
$$\text{accuracy}_{\text{test}} = 0.969$$

For the logistic regression on the normalised data:

The parameters of the model were:

[-0.15301671, 0.41886541, 1.20183536, 0.73123418,
0.47797006, 1.60814454, -0.9043635, -1.69624452,
-3.2546309, -2.55718762, -1.45943251, -0.27085139,
-0.80955427]

with bias:

$$-3.10513732$$

The zero-one loss on the training data was: 0.0300, and on the test data: 0.0383.

For SVMs, the accuracy improved marginally on the normalised training data, but stayed constant for the normalised test data. Since the choice of kernel uses the squared Euclidian distance $\left(||x - x'||^2\right)$, this places a different level of importance on different features. In the training data, the ranges of the different features spanned from 112 for feature 1, to 7205 for feature 4. This squared Euclidian distance measure means that a difference of 1 in the 4th feature (a tiny amount of difference, relative to the range is equivalent to, about 0.01%) would be equivalent to a difference of approximately 0.90% in the 1st feature. This shows that small differences are exaggerated in feature 4 from the Euclidian distance calculation. By normalising the data, these differences in ranges are evened out to adjust for this problem. In the transformed data, the range of feature 1 is 10.78, and feature 4 is 4.80, which means distance in both is much more relative. The results also converged faster than without the data normalisation.

However, for Logistic Regression, the loss increased for the training data samples, but decreased (by a very, very marginal amount, 0.001) for the test data. The normalisation did not have as much of an effect as with the SVMs. This is due to the fact that the algorithm looks at the proportional relationships between the coefficients, which doesn't change with normalisation, so it would be expected to perform similarly to the non-normalised data.

With random forests, normalisation does not affect the performance. This is as in the algorithm, the magnitude of one feature is never directly compared to another feature - only one feature is split at each stage. Convergence and precision don't affect the performance of the random forest algorithm. The formal way of describing this is by saying that random forests are invariant to monotonic transformations of individual features.

2.4 Principal Component Analysis

The PCA was performed using a combination of the `sklearn.decomposition.PCA` package in Python, and previous code I had written for an assignment. The package uses Singular Value Decomposition to project the data to a lower dimension.

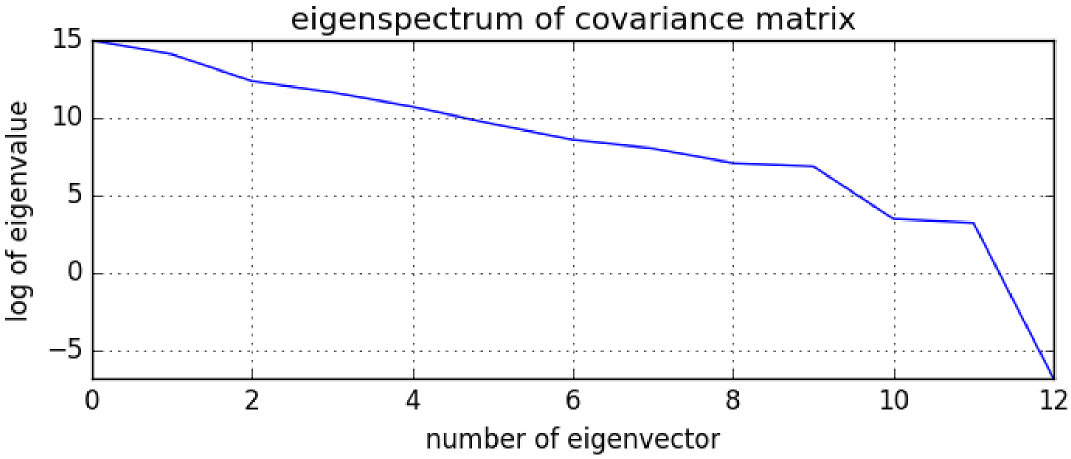

Figure 1: Eigenspectrum of the data

The PCA gave 13 principal components, of which only 2 were necessary to explain 90% of the variance. This is verified by looking at the eigenspectrum plot in Figure 1. This plots the log of the eigenvalues against the principal component number.

The scatter plot in Figure 2 shows the data projected onto the first two principal components of the resulting 13. The legend can be used to identify the class of the points.

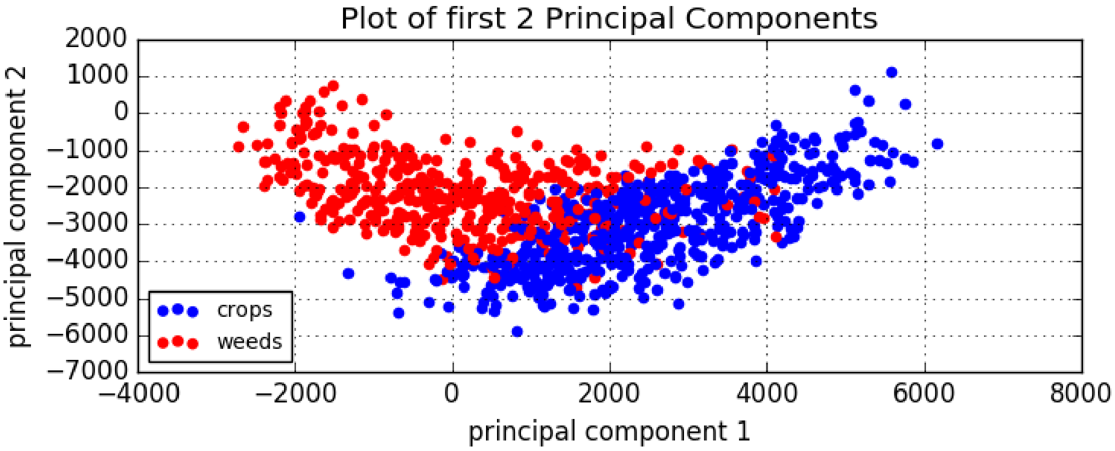

Figure 2: Plot of the data projected onto the first two principal components

Looking at the plot, one can see how important these first 2 principal components are, as visually the clusters are mainly separated by their colours. In comparison, a plot of the data projected against the final 2 principal components is mostly concentrated around the center.

2.5 Clustering

The K-Means Algorithm was implemented using a combination of the `sklearn.cluster.KMeans` package in Python, and previous code I had written for an assignment. The K-Means implementation was used with $k = 2$. The standard full K-Means algorithm was used with a maximum of 300 iterations. The cluster start points were initialised to the first 2 points in the training set.

The projection of the resulting cluster centers onto the first two principal components can be in Figure 3. From the plot, the K-Means algorithm performed very well on the data projected onto the first 2 principal components, as by-eye, each cluster center sits approximately in the center of each of the crops and weeds clusters. The algorithm classifies by measuring the distance between the test point and the two cluster centers, and returning the class of the closest cluster center.

The cluster centers were projected onto the first two principal components by taking the matrix of the cluster centers and performing the dot product of it against each principal component vector. This results in the projected clusters on each of the principal component vectors.

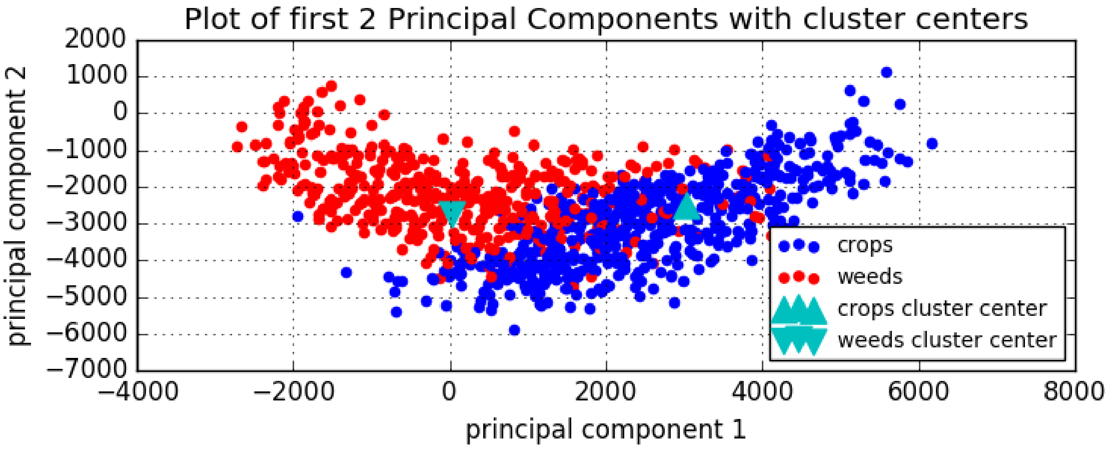

Figure 3: Plot of the data projected onto the first two principal components, with cluster centers from 2-Means clustering

3 Generalization Bound for Learning with Multiple Feature Mappings

3.1

Note that a $\tilde{d}$-dimensional linear separator has $d_{VC} = \tilde{d} + 1$ [2, p. 52], where $\tilde{d}$ is the dimensionality of the transformed data $\in \mathcal{H}_Q$ by $\Phi_Q(\mathbf{x})$.

In calculating the dimensionality of the new space, the different combinations of the original $(x_1 ... x_d)$ points to form the up to order Q monomials were considered. An upper bound may also be placed on the dimension of $\Phi_Q(\mathbf{x})$ by the dimension of $\Phi_Q^+(\mathbf{x})$, as this includes the repetition of identical terms, and therefore will be higher than $\Phi_Q(\mathbf{x})$. To this end, if the terms are considered in order from Q = 0 up to Q, there are d^Q combinations of each order monomial, such as:

$$\begin{aligned} Q &= 0, \quad 1 &&= d^0 = 1 \\ Q &= 1, \quad x_1, x_2, ..., x_d &&= d^1 = d \\ Q &= 2, \quad x_1^2, x_1x_2, ..., x_d^2 &&= d^2 \\ &\text{etc.} \end{aligned}$$

Which results in $\tilde{d}$ equal to:

$$\sum_{i=0}^{Q} d^i$$

and therefore d_{VC} equal to:

$$(\sum_{i=0}^{Q} d^i) + 1$$

There are $(Q + 1)$ terms in this sum, with the highest order polynomial equal to d^Q, which means that an upper bound can be placed on this, and:

$$d_{VC}(\mathcal{H}_Q) \leq (Q + 1) \cdot d^Q$$

3.2

Using the bound in equation 2.14 [2, p. 58], and substituting in the value for the d_{VC}, to get the following bound, with probability at least $1 - \delta, \forall_{h \in \mathcal{H}_Q}$ (N.B. here, and from now on, I am using the derivations from the book, i.e. from the two-sided Hoeffding):

$$L(h) \leq \hat{L}(h, S) + \sqrt{\frac{8}{N} \ln \left(\frac{4 \cdot ((2N)^{(Q+1) \cdot d^Q} + 1)}{\delta} \right)}$$

3.3

If the polynomial kernel with the 2nd order transform is considered:

$$\Phi_2(\mathbf{x}) = (1, x_1, x_2, ...x_d, x_d)$$

and then the product:

$$\Phi_2(\mathbf{x})^T \Phi_2(\mathbf{x}') = 1 + \sum_{i=1}^{d} x_i x_i' + \sum_{i=1}^{d} x_i x_j x_i' x_j'$$

which is equivalent to:

$$1 + (\mathbf{x}^T \mathbf{x}') + (\mathbf{x}^T \mathbf{x}')^2$$

[2, p. 8-34]

For the Qth order transform, this product equals:

$$\Phi_Q(\mathbf{x})^T \Phi_Q(\mathbf{x}') = 1 + (\mathbf{x}^T \mathbf{x}') + (\mathbf{x}^T \mathbf{x}')^2 + ...(\mathbf{x}^T \mathbf{x}')^Q = \sum_{i=0}^{Q} (\mathbf{x}^T \mathbf{x}')^i$$

An upper bound on this can be placed as the Qth polynomial kernel, which equals:

$$\sum_{i=0}^{Q} (\mathbf{x}^T \mathbf{x}')^i \leq (1 + \mathbf{x}^T \mathbf{x})^Q$$

as it includes all the other terms from the sum, just with higher coefficients.

Note that I may bound $\Phi_Q(\mathbf{x})$ by $\Phi_Q^+(\mathbf{x})$, as it includes more terms due to the repeated identical terms, and if I set the repeated terms equal to 0, they are equivalent.

From the above, consider that:

$$\begin{aligned}
||\mathbf{x}|| &\leq 1 \\
\sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} &\leq 1 \\
\langle \mathbf{x}, \mathbf{x} \rangle &\leq 1 \\
\mathbf{x}^T\mathbf{x} &\leq 1 \\
\mathbf{x}^T\mathbf{x} + 1 &\leq 2 \\
(\mathbf{x}^T\mathbf{x} + 1)^Q &\leq 2^Q \\
(\Phi_Q^+)T(\Phi_Q^+) \leq (\mathbf{x}^T\mathbf{x} + 1)^Q &\leq 2^Q \\
\sqrt{\Phi_Q^+(\mathbf{x})^T\Phi_Q^+(\mathbf{x})} \leq \sqrt{(\mathbf{x}^T\mathbf{x} + 1)^Q} &\leq \sqrt{2^Q} \\
\sqrt{\langle \Phi_Q^+(\mathbf{x}), \Phi_Q^+(\mathbf{x}) \rangle} \leq \sqrt{(\mathbf{x}^T\mathbf{x} + 1)^Q} &\leq \sqrt{2^Q} \\
||\Phi_Q(\mathbf{x})|| \leq \sqrt{(\mathbf{x}^T\mathbf{x} + 1)^Q} &\leq \sqrt{2^Q}
\end{aligned}$$

3.4

Consider that:

$$||\Phi_Q(\mathbf{x})|| \leq \sqrt{2^Q} \in \mathbb{R}^{\tilde{d}}$$

from the above exercise.

Using an adjusted version of Theorem 3.8 [3, p. 18]

$$d_{VC}(\mathcal{H}_\rho) \leq \lceil {}^{R^2}\!/_{\rho^2} \rceil$$

I adjusted this as I am considering the space of hyperplanes: $\mathcal{H}_\rho = \{(w) : ||\mathbf{w}|| \leq {}^1\!/_\rho\}$. However, the input space is in $\mathbb{R}^{\tilde{d}}$ defined by $\Phi_Q(\mathbf{x})$, so from the previous answer, as $\mathbf{x}$ is a ball of radius 1, in the tranformed space we have that $R = \sqrt{2^Q}$.

Plugging this into the above result yields:

$$d_{VC}(\mathcal{H}_\rho) \leq \lceil {}^{\left(\sqrt{2^Q}\right)^2}\!/_{\rho^2} \rceil = \lceil {}^{2^Q}\!/_{\rho^2} \rceil$$

Now, considering $\mathcal{H}$ as a nested sequence of subspaces, $\mathcal{H}_1 \subset \mathcal{H}_2 ... \subset \mathcal{H}_d = \mathcal{H}$, and defining $\mathcal{H}_i = \mathcal{H}_{\rho = 2^Q/i}$, where $i = 2^Q/\rho^2$. Also note that if $h = w \in \mathcal{H}_i \setminus \mathcal{H}_{i-1}$, then $i = \lceil 2^Q \cdot ||\mathbf{w}||^2 \rceil$ as:

$$\begin{aligned} ||\mathbf{w}|| &\leq \frac{1}{\rho} \\ ||\mathbf{w}||^2 &\leq \frac{1}{\rho}^2 \\ 2^Q \cdot ||\mathbf{w}||^2 &\leq \frac{2^Q}{\rho^2} \\ \lceil 2^Q \cdot ||\mathbf{w}||^2 \rceil &= i \end{aligned}$$

I can also take

$$\delta_i = \frac{1}{i} i + 1$$

Using the result from before gives:

$$d_{VC}(\mathcal{H}_i) = i$$

which gives through Theorem 3.7 [3, p.18]:

$$\mathbb{P}\left\{ \exists h \in \mathcal{H}_i : L(h) \geq \hat{L}(h,S) + \sqrt{\frac{8}{n} \cdot \ln \frac{\big(4\big((2n)^i + 1\big)\big)}{\delta_i}} \right\} \leq \delta_i$$

then we plug in and derive:

$$\mathbb{P}\left\{ \exists h \in \mathcal{H}_i : L(h) \geq \hat{L}(h,S) + \sqrt{\frac{8}{n} \cdot \ln \frac{\big(4\big((2n)^{(\lceil 2^Q ||\mathbf{w}|| \rceil)} + 1\big) \cdot \big(((\lceil 2^Q ||\mathbf{w}|| \rceil) \cdot (((\lceil 2^Q ||\mathbf{w}|| \rceil) + 1))\big)}{\delta}} \right\}$$

$$= \mathbb{P}\left\{ \exists h \in \bigcup_{i=1}^{d} \mathcal{H}_i \setminus \mathcal{H}_{i-1} : L(h) \geq \hat{L}(h,S) + \right.$$

$$\left. \sqrt{\frac{8}{n} \cdot \ln \frac{\big(4\big((2n)^{(\lceil 2^Q ||\mathbf{w}|| \rceil)} + 1\big) \cdot \big(((\lceil 2^Q ||\mathbf{w}|| \rceil) \cdot (((\lceil 2^Q ||\mathbf{w}|| \rceil) + 1))\big)}{\delta}} \right\}$$

$$= \sum_{i=1}^{d} \mathbb{P}\left\{ \exists h \in \bigcup_{i=1}^{d} \mathcal{H}_i \setminus \mathcal{H}_{i-1} : L(h) \geq \hat{L}(h,S) + \right.$$

$$\left. \sqrt{\frac{8}{n} \cdot \ln \frac{\big(4\big((2n)^{(\lceil 2^Q ||\mathbf{w}|| \rceil)} + 1\big) \cdot \big(((\lceil 2^Q ||\mathbf{w}|| \rceil) \cdot (((\lceil 2^Q ||\mathbf{w}|| \rceil) + 1))\big)}{\delta}} \right\}$$

$$= \sum_{i=1}^{d} \mathbb{P}\Bigg\{ \exists h \in \mathcal{H}_i \setminus \mathcal{H}_{i-1} : L(h) \geq \hat{L}(h,S) +$$

$$\sqrt{\frac{8}{n} \cdot \ln \frac{\big(4\big((2n)^{(\lceil 2^Q||\mathbf{w}||\rceil)} + 1\big) \cdot \big(((\lceil 2^Q||\mathbf{w}||\rceil) \cdot ((\lceil 2^Q||\mathbf{w}||\rceil) + 1))\big)}{\delta}}\Bigg\}$$

$$= \sum_{i=1}^{d} \mathbb{P}\Bigg\{ \exists h \in \mathcal{H}_i \setminus \mathcal{H}_{i-1} : L(h) \geq \hat{L}(h,S) + \sqrt{\frac{8}{n} \cdot \ln \frac{\big(4\big((2n)^i + 1\big) \cdot \big(i \cdot (i+1)\big)\big)}{\delta}}\Bigg\}$$

$$\leq \sum_{i=1}^{d} \mathbb{P}\Bigg\{ \exists h \in \mathcal{H}_i : L(h) \geq \hat{L}(h,S) + \sqrt{\frac{8}{n} \cdot \ln \frac{\big(4\big((2n)^i + 1\big) \cdot \big(i \cdot (i+1)\big)\big)}{\delta}}\Bigg\}$$

Then using Hoeffding's

$$\leq \sum_{i=1}^{d} \delta_i = \sum_{i=1}^{d} \frac{1}{i \cdot (i+1)} = \delta \sum_{i=1}^{d} \frac{1}{i \cdot (i+1)} \leq \delta \sum_{i=1}^{\infty} \frac{1}{i \cdot (i+1)} = \delta$$

This model of proof was taken from the lecture notes. [3, p. 19]

Therefore, the bound on $h \in \mathcal{H}$, w.p. at least $1 - \delta$ is:

$$L(h) \leq \hat{L}(h,S) + \sqrt{\frac{8}{n} \cdot \ln \frac{\big(4\big((2n)^{(\lceil 2^Q||\mathbf{w}||\rceil)} + 1\big) \cdot \big(((\lceil 2^Q||\mathbf{w}||\rceil) \cdot ((\lceil 2^Q||\mathbf{w}||\rceil) + 1))\big)}{\delta}}$$

3.5

The VC dimension is defined as the largest N such that $m_{\mathcal{H}}(N) = 2^N$. It is also true that $m_{\mathcal{H}} \leq \min\left(|\mathcal{H}|, 2^N\right)$ [4, Home Assignment 5.2.1]. Finally, if $m_{\mathcal{H}}(N) = 2^N \ \forall_N, d_{VC} = \infty$. [2, def. 2.5, p.50]

As $|\mathcal{H}|$ is infinite for the union of all polynomial transformations, for all N, $m_{\mathcal{H}} = 2^N$, and following the definition above, this implies that $d_{VC} = \infty$.

3.6

References

[1] Wikipedia, *Coordinate descent - wikipedia the free encyclopedia*, [Online; accessed 3-November-2016], 2016. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Coordinate_descent&oldid=747699222`.

[2] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data*. AML-Book Singapore, 2012, vol. 4.

[3] Y. Seldin, *Yevgeny's Lecture Notes 1-Dec-2016*. 2016, vol. 1.

[4] Y. Seldin and C. Igel, *Machine Learning Assignments, KU*. 2016, vol. 5.