

# Machine Learning Final Exam

Department of Computer Science, University of Copenhagen

Dhruv Chauhan

January 20, 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 places).

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]
```

\*\*\* CHECK AND MAYBE PLUG THESE INTO THE MODEL \*\*\*

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,  $\sigma_{red}^2$  were equal to 0.176 for both the test and the training data.

This normalised error falling below one signifies that...

### 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 we 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.

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 we 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. We are 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. (Wikipedia, The Free Encyclopedia, 2016)

The parameters produced by the model were:

$$\begin{aligned} &[-0.0391283, 0.01549887, 0.00295803, 0.00033714, \\ &-0.00039954, 0.00348062, -0.00715483, 0.00473467, \\ &-0.02685346, -0.05592747, -0.04317431, 0.00579407, \\ &-0.00998736] \end{aligned}$$

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, specifically 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:

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

Using the formula given, I calculated:

$$\begin{aligned}\sigma_{\text{Jaakkola}} &= 609 \\ \gamma_{\text{Jaakkola}} &= 1.35e - 06\end{aligned}$$

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

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

$$\begin{aligned}C &= 1000 \\ \gamma &= 1.35e - 8\end{aligned}$$

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

$$\begin{aligned}\text{accuracy}_{\text{training}} &= 0.978 \\ \text{accuracy}_{\text{test}} &= 0.969\end{aligned}$$

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

## 2.3 Normalisation

The normalisation was performed by