COMP3223: Coursework

1 Introduction

1.1 Due date

The hand-in date for the assignment is **Thursday , December 10, 2020** .

1.2 General remarks

The exercises are meant to help you understand the course material in depth. You will find that using tools such as `scikit-learn`, which are there for you to use in practical contexts, will make the task of completing your coursework much simpler. However, for developing a thorough understanding of those implementations you will need to explore how the data gets transformed using mathematical methods, and you need to implement the algorithms yourself. You may use the `scikit-learn` libraries to check your implementations if you wish. However, for many of the data preparation steps such as loading, splitting data into training and test tests randomly and so on, you can use canned routines such as those in `scikit-learn`. For some tasks I have also given you pieces of code that you can reuse, and you can repurpose the `jupyter` lab notebooks as well.

There are umpteen references on the web you may wish to consult. However, for most of the exercises [1], [2] and [3] should give you food for thought and influence the way you write your report.

Some of you may wish to work on other datasets for more challenging tasks than those required of you here. Here are a couple of sources:

1. `https://www.kaggle.com`

2. `https://archive.ics.uci.edu/ml/datasets.php`

1.3 Submission guidelines

The report should focus on the key conceptual steps that underpin each algorithm, illustrating your understanding by pointing to the results you obtain that should be summarised in graphical or tabular form. Some of these are asked for explicitly in the questions below, others are left to your judgement. All else being the same, a well articulated report that shows evidence of depth of understanding will get a higher mark than one where the same results are described, but less cogently. There are page limits signposted in some of the sections and the other sections should merit a similar allocation of space. These limits are not going to be policed rigorously but are intended as a guide for you to gauge how much is too much or too little. There will be a handin page for you to submit a report and code files.

2 Linear Regression with non-linear functions

[4 marks]

In this exercise you have to perform the task of fitting polynomial functions to $y(x) = \sin(4\pi x) + \epsilon$ where ϵ is a noise term. You have to generate a number N of points ($N \approx 15$) chosen randomly from $0 \leqslant x \leqslant 1$. Your training and test sets will be taken from these (x_n, y_n) pairs, for $n = 1, \ldots, N$.

```
rn = np.random.uniform(0, 1, n)
x = np.reshape(np.sort(rn, axis=0), (n, 1))
```

2.1 Performing linear regression with polynomials and radial basis functions

In this part you will model each (input, target) pair (t_n, x_n) by a linear combination of basis functions: $y(x; \boldsymbol{w}) = \sum_j w_j \phi_j(x)$. You will take **two** classes of functions $\phi_j(x)$ for your basis elements to fit the training and test data sets that you generate:

1. polynomials: $\phi_j(x_n) = x_n^j$ where x_n is an input value.
2. Gaussian radial basis functions (RBF): for each training input x_n,

$$\Phi(x; x_n) = \exp\left(-\frac{(x - x_n)^2}{2s^2}\right),$$

and $\phi_j(x_n) = \Phi(x_j; x_n)$ for $0 \leqslant x_j \leqslant 1, j = 1, \ldots, p$.

1. Construct a design matrix $\mathbf{A}$ by concatenating to a column of ones p columns $\phi_1(x_n), \ldots, \phi_p(x_n)$. The column entries are $\phi_j(x_n), n = 1, \ldots, N$.

```
def gaussian_basis_fn(x, mu, sigma=0.1):
    return np.exp(-0.5 * (x - mu) ** 2 / sigma ** 2)

def polynomial_basis_fn(x, degree):
    return x ** degree

 def make_design(x, basisfn, basisfn_locs=None):
    if basisfn_locs is None:
        return np.concatenate([np.ones(x.shape), basisfn(x)], axis=1)
    else:
        return np.concatenate([np.ones(x.shape)] + \
        [basisfn(x, loc) for loc in basisfn_locs], axis=1)
```

2. The weights that minimise the loss function

$$\sum_{n=1}^{N} \left(y_n - w_0 - \sum_{j=1}^{p} w_j \phi_j(x_n) \right)^2 + \lambda \|\boldsymbol{w}\|_2^2$$

are given by the analytical expression[1]

$$\boldsymbol{w} = (\mathbf{A}^T \mathbf{A} + \lambda \mathbb{I}_{p+1})^{-1} \mathbf{A}^T \mathbf{y}, \tag{1}$$

where $\mathbb{I}_m$ stands for a $m \times m$ identity matrix[2].

2.2 Generalisation

[3 marks]

This section gets you to explore the models learned by linear regression, by looking at the dependence of the optimal weights on training data and the scale of regularisation. When writing up the results of this section about the ability of the models to generalise using the tasks below, try to relate the weights from eq.(1) to your analysis.

[1] You should know how to derive it.

[2] An alternative expression is $\boldsymbol{w} = \mathbf{A}^T (\mathbf{A}\mathbf{A}^T + \lambda \mathbb{I}_N)^{-1} \mathbf{y}$

1. This is about generalisation performance as a function of complexity and the effect of regularisation on the same task as above.

   a: Generate $N \approx 25$ points and split them into a training $\mathcal{S}^{tr}$ and test set $\mathcal{S}^{ts}$. You could call

   ```
   from sklearn.model_selection import train_test_split
   ```

   and look up the function description of `train_test_split`.

   b: Measure the mean of the squared residuals on the test set $\mathcal{S}^{ts}$ as a performance metric for each model. Plot this measure as a function of:

   i. the number of basis components ϕ_j ;

   ii. the strength of the regularisation coefficient λ.

   c: How do these measures depend on the choice of training/test splits?

3 Classification

You will explore two methods for some toy datasets in order to get familiar with some of the issues to do with projections and data transformations. You will generate one of the two datasets yourself using a random number generator. The other is an old classic – Ronald Fisher's Iris dataset that can be loaded using `scikit-learn`'s data loader.

You will first explore Fisher's LDA for binary classification for class labels a and b. In Fisher's method a direction defined by vector $\boldsymbol{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$ is chosen so that the data $\boldsymbol{x}^n$ projected on to $\boldsymbol{w}$ maximises the separation between the a and b type distributions. (n is a data index, ranging from 1 to n_a for class a, and from 1 to n_b for class b, here used in superscript; it is not the n-th power of $\boldsymbol{x}$.) Setting $y_c^n \triangleq \boldsymbol{w} \cdot \boldsymbol{x}_c^n$ for class label $c \in \{a, b\}$ the scalar means and standard deviations of the projected data become

$$\mu_c = \frac{1}{n_c} \sum_{n=1}^{n_c} y_c^n, \ \sigma_c^2 = \frac{1}{n_c} \sum_{n=1}^{n_c} (y_c^n - \mu_c)^2, \ c \in \{a, b\}.$$

The direction $\boldsymbol{w}$ is chosen in order to maximise the Fisher ratio $F(\boldsymbol{w})$: Fisher ratio

$$F(\boldsymbol{w}) \triangleq \frac{(\mu_a - \mu_b)^2}{\frac{n_a}{n_a + n_b}\sigma_a^2 + \frac{n_b}{n_a + n_b}\sigma_b^2}.$$

3.1 Data 1: separate 2 Gaussians

The data for this part are to be generated by you. Generate data from two 2-dimensional Gaussian distributions

$$\mathbf{x}_a \sim \mathcal{N}(\mathbf{x}|\mathbf{m}_a, \mathbf{S}_a),\ \mathbf{x}_b \sim \mathcal{N}(\mathbf{x}|\mathbf{m}_b, \mathbf{S}_b)$$

where $\mathbf{m}_a, \mathbf{m}_b$ are the (2×1) mean vectors and $\mathbf{S}_a$ and $\mathbf{S}_b$ are the (2×2) covariance matrices that define normal distributions. Let the number of data points from each type a, b be n_a and n_b. If the classes are widely separated or if they overlap significantly it will be hard for you to explore and demonstrate the consequences of choosing different directions, and thus learn from the experience. Hence, make the differences of the means of the order of magnitude of the standard deviations. The precise values will affect the results of the following tasks and you can experiment with numerical values that help you appreciate the pros and cons of the classification method. This will determine the evidence you provide in the report that demonstrate your insights on the nature of the learning algorithm.

The following tasks are broken up into two chunks to demarcate how the marks are to be allotted, even though they are all connected. For reference, you should read Section 5.2 of FCML [1], Section 4.2 of Bishop [2], Section 4.4 of [4] or Section 4.3 of [3] for guidance and ideas.

1. This part is for visual exploration of the consequences of projecting data onto a lower dimension.

   [3 marks]

   (a) Make *two* illustrative choices for the direction $\boldsymbol{w}$ and plot the histograms of the values y_a^n and y_b^n. You should make choices that make a difference to the nature of the resulting histograms.

   (b) Alter the values of $\mathbf{S}_a$ and $\mathbf{S}_b$ to show how the histograms in the previous task change. Choose $2-4$ matrices to demonstrate how class separation is affected.

   (c) Plot the dependence of the Fisher ratio $F(\boldsymbol{w})$ on the direction of $\boldsymbol{w}$ by rotating some random starting weight vector $\boldsymbol{w}(0) = (w_1, w_2)^T$ and rotating it by angles θ to get $\boldsymbol{w}(\theta) = R(\theta)\boldsymbol{w}(0)$ where

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

Find the maximum value of $F(w(\theta))$ and the corresponding direction w^*:

$$w^* = \underset{w}{\operatorname{argmax}}\, F(w) = \underset{\theta}{\operatorname{argmax}}\, F(w(\theta)).$$

Aside: In the lectures you will have come across the idea of change of basis using orthogonal transformations, for instance, using sets of orthogonal eigenvectors/singular vectors. The columns of the matrix $R(\theta)$ are orthogonal.

2. In this part you will work with both the known parameters of the mean and covariances you have chosen as well as their maximum likelihood estimates (MLE) from the data you have generated.

[4 marks]

   (a) Use equations (5.4) and (5.5) of FCML [1] to estimate the means and covariances of the data generating distributions.

   (b) Use Bayes' theorem to write down the logarithm of the ratio of conditional probabilities (called log-odds)

$$\ln\left(\frac{P(c = a|x^n)}{P(c = b|x^n)}\right)$$

   and plot the decision boundary for both $S_a = S_b$ and $S_a \neq S_b$. In other words, for each point $x \in \mathbb{R}^2$ check whether the log-odds is positive, negative or zero. The decision boundary is the set of points where the log-odds vanishes. Do this for the *true* parameters as well as the *estimated* parameters from MLE. Comment on the results you obtain.

3.2 Data 2: Iris data

[6 marks]

In this section you will perform the same LDA task on the famous Iris dataset `https://en.wikipedia.org/wiki/Iris_flower_data_set` which can be downloaded from `http://archive.ics.uci.edu/ml/datasets/Iris`. While the previous exercise was done by finding the weight vector to project on by

direct search, here you follow Fisher and solve the generalised eigenvalue condition for optimal weights. For an introduction to the method, read Section 4.1.4 of Bishop [2].

With more features and classes, you will need to compute the between-class and within-class covariance matrices Σ_B and Σ_W:

$$\Sigma_B = \sum_c \frac{n_c}{N}(\mu_c - \mu)(\mu_c - \mu)^T, \text{where } \mu \text{ is the mean of class means,}$$

and Σ_W is the sum of the covariance matrices for each class. n_c is the number of data samples in class c and $N = \sum_c n_c$. In case there are different numbers of training data points from each class, you have to scale any class dependence by the corresponding fraction of class members in the population. (In the Iris data set all three classes have 50 members, so you can skip this step.)

1. Find the optimal direction w^* for projecting the data onto. You will need to solve the generalised eigenvalue problem $\Sigma_B w = \lambda \Sigma_W w$. Section 16.2, 16.3 of [5] and eq. (4.15) of [3] has further details.

   Suggestions: *The standard libraries in scipy (and others) can give you the generalised eigenvalues and eigenvectors. Since the covariance matrix is symmetric, the function you should call in* `numpy/scipy` *is* `eigh`*, and not* `eig`*, although for problems of this scale it won't make a difference and you can eyeball the eigenvalues. In particular, the eigenvalues returned by* `eigh` *are* sorted. You must also check the answer provided by verifying that the generalised eigenvalue condition $\Sigma_B w = \lambda \Sigma_W w$ holds. This will clarify the notational conventions of the software used. Sometimes it is the transpose of the returned matrix of vectors that contains the eigenvectors, so please make sure you understand what is being returned. *You should also discover that the rank of Σ_B is limited by the number of classes.* Rank condition

2. Display the histograms of the three classes in the reduced dimensional space defined by the eigenvectors.

3. This is for using softmax regression on the same task.

   a: Perform softmax regression on the same dataset to classify the points.

   b: Since the weight vectors of the 3 classes in softmax regression are not independent (they are constrained by the probability requirement) find a way of representing the data in a 2-dimensional projection using the learned weights. (See eq. (4.17) of [3].)

4. Present your results with reflections and evidence, comparing all three methods.

References

[1] Simon Rogers and Mark Girolami. *A First Course in Machine Learning, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2016.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2nd edition, 2009.

[4] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

[5] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 04-2011 edition, 2011.