Machine Learning and Pattern Recognition - Week 3 - Classification & Gradient Based Fitting

Antonio León Villares

October 2022

Contents

1 Classification by Regression **2**
1.1 The Task of Classification . . . 2
1.2 Challenges of Linear Regression for Classification . . . 2
1.3 One-Hot Encodings . . . 4
1.4 Computing Model Parameters . . . 4
1.4.1 Closed-Form Solution . . . 4
1.4.2 Iterative Methods: Gradient Descent . . . 6

2 Classification by Bayesian Models **8**
2.1 Multivariate Gaussian . . . 8
2.2 Example: Gaussians for Cancer . . . 8
2.3 Naive Bayes . . . 10

3 Classification by Logistic Regression **10**

4 Questions **12**
4.1 Notes Questions . . . 12

5 Tutorial **13**

Based on the online notes here.

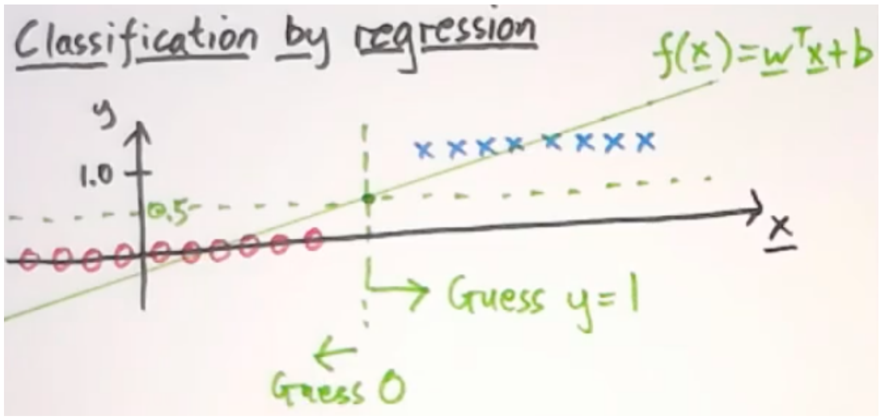
1 Classification by Regression

1.1 The Task of Classification

- **What is classification?**
  - given some **inputs** $\underline{x}^n$, predict the label $y^{(n)}$
  - here, the $y^{(n)}$ come from a **discrete** set of **class labels**
- **What is binary classification?**
  - classifying $\underline{x}^n$ into one of 2 classes
  - typically use $\{0, 1\}$ or $\{-1, 1\}$
  - for instance, determining if an email is **spam**, or if a certain feature appears in an image
- **What is multiclass classification?**
  - classifying $\underline{x}^{(n)}$ into more than 2 classes
  - for example, the genre of a novel, or the type of sport played by a student
- **What is multilabel classification?**
  - classifying $\underline{x}^{(n)}$ into more than one class
  - for example, a show defined as a romantic-comedy has the class “romance” and “comedy”

1.2 Challenges of Linear Regression for Classification

- **How can linear regression be applied to classification?**
  - we can train a function $f(\underline{x})$, such that (for binary classification) if $f(\underline{x}) > \frac{1}{2}$, we output class 1, and class 0 otherwise



- **What are the issues associated with using regression for classification?**
  1. **A class can extend over multiple values**
     - for example, for $x \in [0, 1] \cup [5, 9]$, we can have class 0, and for $x \in (1, 5)$ we can have class 1
     - a **naive** linear regression wouldn’t be able to adapt to this

- even if we use basis functions, the **least squares** fit will attempt to overfit to one of the classes

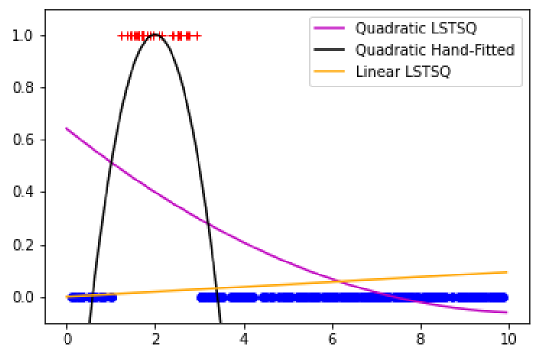


Figure 1: The black quadratic makes a good split of the data, so $f(\underline{x}) > 0.5$ would be a good classifier. The magenta quadratic tries to fit itself to the large number of negative labels on the RHS, leading to a bad fit. The orange linear is hopeless in this case, since the presence of even a single positive label will make it so that it has a gradient, and so, will be a bad classifier.

2. **The same input can have 2 different outputs**
   - labels can be noisy, so it is possible that the same instance can have 2 classes assigned
   - this artifically increases the loss
   - a different loss than **least squares** can be used, based on **expectation**:

$$\begin{aligned}\mathbb{E}[(y-f)^2] &= p_1(1-f)^2 + (1-p_1)(0-f)^2 \\ &= f^2 - 2p_1 f + p_1\end{aligned}$$

     where $p_1 = p(y = 1 \mid \underline{x})$ and $p_0 = p(y = 0 \mid \underline{x}) = 1 - p_1$
   - minimising by differentiating with respect to f leads to:

$$f = p_1$$

   - that is, the best function $f = f(\underline{x})$ is that such that:

$$f(\underline{x}) = p(y = 1 \mid \underline{x})$$

     so for f to minimise cost, it should return the probability of an input belonging to class 1
   - this can be attained by using a large amount of **basis functions** (i.e RBFs), which will produce a very flexible regression model
   - however, this interpretation is still flawed, since f will extend outside the range $[0, 1]$
3. **Hard to Interpret for Multiclass Classification**
   - say we have the following classes:

$$\{1 = \text{“}sport\text{”}, 2 = \text{“}crime\text{”}, 3 = \text{“}technology\text{”}\}$$

     for article classification
   - then, an average of inputs which are classified as “sport” and “technology” would be classified as “crime”, which isn't a useful interpretation:

$$f(\underline{x}^{(1)}) = \underline{w}^T \underline{\phi}(\underline{x}^{(1)}) \approx 1 \qquad f(\underline{x}^{(3)}) = \underline{w}^T \underline{\phi}(\underline{x}^{(1)}) \approx 3 \implies \underline{w}^T \left[\underline{\phi}(\underline{x}^{(1)}) + \underline{\phi}(\underline{x}^{(3)})\right] \approx \frac{1+3}{2} = 2$$

1.3 One-Hot Encodings

- **What are one-hot encodings?**
  - a way of numerically encoding **categorical** labels
  - given a set of K classes, we could encode class k as a **binary** vector $\underline{v}_k \in \mathbb{R}^K$, with a 1 at position k, and 0s elsewhere
- **Why are one-hot encodings used for regression?**
  - they remove the **interpretability** issues, since the "average" of 2 vectors assigned to different classes won't immediately lead to its classification as a third class:

$$0.5 \times \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + 0.5 \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = 0.5 \times \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

- **How can linear regression be fit to predict one-hot vectors?**
  - we can create a model for each component of the one-hot vectors:

$$\begin{pmatrix} f_1(\underline{x}) \\ f_2(\underline{x}) \\ \vdots \\ f_K(\underline{x}) \end{pmatrix}$$

  - we would then predict the class of $\underline{x}$ as the component with the largest value
- **How can one-hot vectors be used for multilabel classification?**
  - a multilabel class can be given as a sum of the one-hot encodings of its individual categories
  - we'd fit linear regression as specified above, and select the top C components
- **Can we use one-hot encodings for inputs aswell?**
  - yes; for example, we can encode certain features (i.e colour) by using a one-hot encoding
  - intuitive, since we shouldn't assume that inputs of different colours should be related

1.4 Computing Model Parameters

1.4.1 Closed-Form Solution

Given training data X, the set of weights minimising the ***least squares error*** *of:*

$$\underline{y} \approx \hat{\underline{y}} = X\underline{w}$$

is given by:

$$\underline{w} = (X^T X)^{-1} X^T \underline{y}$$

where $(X^T X)^{-1} X^T$ is the ***pseudo-inverse****, assuming that $(X^T X)^{-1}$ exists.*

Proof. Let $\underline{r}$ denote the residuals of our model:

$$r = \underline{y} - X\underline{w}$$

Then, the least squares error expression is:

$$\underline{r}^T\underline{r} = (\underline{y} - X\underline{w})^T(\underline{y} - X\underline{w}) = \underline{y^T y} - 2\underline{w}^T X^T \underline{y} + \underline{w}^T X^T X\underline{w}$$

1. **Matching Coefficients**

   A general quadratic expression in $\underline{w}$ is given by:

   $$(\underline{w} - \underline{\hat{w}})^T M(\underline{w} - \underline{\hat{w}}) + C = \underline{w}^T M\underline{w} - 2\underline{w}^T M\underline{\hat{w}} + C$$

   where $\underline{\hat{w}}$ is the extremum (**turning point**) of the quadratic. In other words, finding an expression for $\underline{\hat{w}}$ will give us the minimum of our least squares expression.

   To do this, we can match the coefficients of our least squares with those of the general quadratic. It is immediately clear that:

   $$2\underline{w}^T M\underline{\hat{w}} = 2\underline{w}^T X^T\underline{y} \implies M\underline{\hat{w}} = X^T\underline{y}$$

   Moreover,

   $$\underline{w}^T M\underline{w} = \underline{w}^T X^T X\underline{w} \implies M = X^T X$$

   Hence, assuming that M^{-1} exists:

   $$\underline{\hat{w}} = M^{-1}X^T\underline{y} = (X^T X)^{-1}X^T\underline{y}$$

2. **Calculus**

   We can consider extremising least squares with respect to $\underline{w}$. That is, we want to compute:

   $$\nabla_{\underline{w}}[\underline{r}^T\underline{r}]$$

   The components of $\nabla_{\underline{w}}[\underline{r}^T\underline{r}]$ are:

   $$\frac{\partial(\underline{r}^T\underline{r})}{\partial w_i} = \frac{\partial(\underline{y}^T\underline{y})}{\partial w_i} - 2\frac{\partial(\underline{w}^T X^T\underline{y})}{\partial w_i} + \frac{\partial(\underline{w}^T X^T X\underline{w})}{\partial w_i}$$

   Now:

   $$\frac{\partial(\underline{x}^T\underline{v})}{\partial x_i} = \sum_j \frac{\partial(x_j v_j)}{x_i} = v_j$$

   $$\frac{\partial(\underline{x}^T A\underline{x})}{\partial x_i} = \sum_j\sum_j x_j A_{jk}x_k = \sum_k A_{ik}x_k + \sum_j x_j A_{ji}$$

   Hence:

   $$\frac{\partial(\underline{r}^T\underline{r})}{\partial w_i} = \frac{\partial(\underline{y}^T\underline{y})}{\partial w_i} - 2\frac{\partial(\underline{w}^T X^T\underline{y})}{\partial w_i} + \frac{\partial(\underline{w}^T X^T X\underline{w})}{\partial w_i} = -2(X^T y)_i + \sum_k (X^T X)_{ik}x_k + \sum_j x_j(X^T X)_{ji}$$

   But notice, $X^T X$ is symmetric, so:

   $$\frac{\partial(\underline{r}^T\underline{r})}{\partial w_i} = -2(X^T y)_i + 2\sum_k (X^T X)_{ik}x_k$$

Thus the gradient becomes:

$$\nabla_{\underline{w}}[\underline{r}^T\underline{r}] = -2X^T\underline{y} + 2X^TX\underline{w}$$

If we set this to 0, then:

$$\begin{aligned}&\nabla_{\underline{w}}[\underline{r}^T\underline{r}] = \underline{0}\\ \implies &X^TX\underline{w} = X^T\underline{y}\\ \implies &\underline{w} = (X^TX)^{-1}X^T\underline{y}\end{aligned}$$

The above is known as the **normal equation** solution of the least squares problem.

$\square$

*Whilst we have a **closed-form** solution, it is **bad practice** to compute X^TX, and then invert it. This is because we will require an algorithm, which comes with its issues (i.e numerical errors).*
*Instead, we use **numerical solvers** to solve the normal equation:*

```
w = np.linalg.solve(np.dot(X.T, X), np.dot(X.T, y))
```

*It is even better to use **dedicated routings**:*

```
np.lstsq(X,y)
```

These can have some trade-offs (i.e accuracy), but often work well enough.

1.4.2 Iterative Methods: Gradient Descent

- **What is steepest gradient descent?**
  - a technique for model optimisation
  - given a set of **weights** $\underline{w}$ and a **loss function** f, we can **update** the weights via:

$$\underline{w} \leftarrow \underline{w} - \eta\nabla_{\underline{w}}f$$

  - η is the **learning rate**, and defines how much "reliability" we want to assign to the gradient
  - steepest gradient descent is a **general** method, which can be used to optimise many loss functions (not just least squares for linear regression)
- **Why do we subtract the gradient?**
  - ∇f is the direction of **greatest change** in the function f
  - by **subtracting**, we are changing $\underline{w}$ to **minimise** f, our loss
  - if we added instead, we'd be optimising to maximise the loss
- **Why does steepest gradient descent lead to improving weights in linear regression?**
  - for linear regression we have:

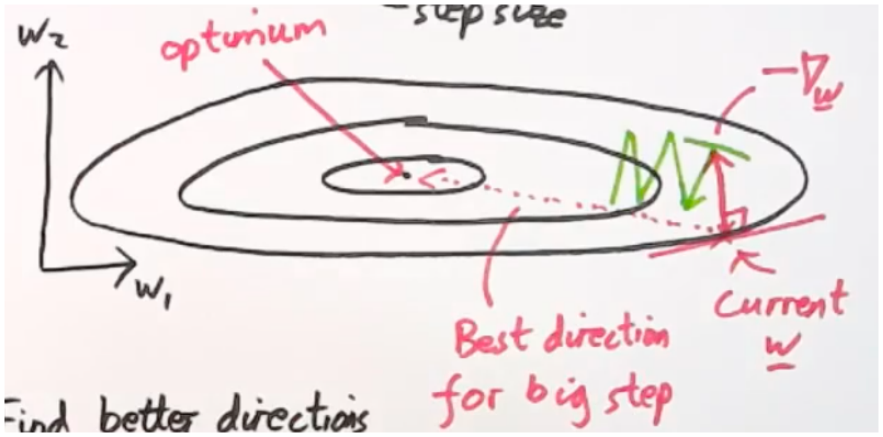
$$\underline{w} \leftarrow \underline{w} - \eta\nabla_{\underline{w}}[\underline{r}^T\underline{r}]$$

- we can rewrite the gradient:

$$\begin{aligned}\nabla_{\underline{w}}[\underline{r}^T\underline{r}] &= 2X^T(X\underline{w}) - 2X^T\underline{y} \\ &= 2X^T(f - \underline{y}) \\ &= 2\sum_{n=1}^{N}\underline{x}^{(n)}(f^{(n)} - y^{(n)})\end{aligned}$$

- hence, weights are being updated so that inputs $\underline{x}^{(n)}$ with the largest residuals reduce their error in the next iteration

- **What are the issues with steepest gradient descent?**
  1. **Convergence Speed**: whilst ∇f is the direction of greatest change, and eventually it will reach an optimum, it doesn't mean it is the most direct way of getting there:



     The algorithm oscillates between many values of $\underline{w}$ in a very inefficient manner.
  2. **Computationally Expensive**: before any update to $\underline{w}$, we have to compute:

$$\nabla_{\underline{w}}[\underline{r}^T\underline{r}] = 2\sum_{n=1}^{N}\underline{x}^{(n)}(f^{(n)} - y^{(n)})$$

     If N is large, this becomes very expensive. Moreover, a small learning rate η means that thousands/millions of operations need to be performed before getting significant progress.

- **What is batch steepest gradient descent?**
  - instead of performing a gradient step after seeing **all** the data, we update after seeing a **batch**, whereby we **approximate** the true gradient via:

$$\nabla_{\underline{w}}[\underline{r}^T\underline{r}] = 2\sum_{n=1}^{N}\underline{x}^{(n)}(f^{(n)} - y^{(n)}) = 2N\frac{1}{N}\sum_{n=1}^{N}\underline{x}^{(n)}(f^{(n)} - y^{(n)}) \approx 2N\frac{1}{B}\sum_{n=1}^{B}\underline{x}^{(n)}(f^{(n)} - y^{(n)})$$

  - if $B << N$, then updates to $\underline{w}$ will be more frequent, but hopefully still accurate enough to allow a faster convergence to the optimum

- **What is stochastic gradient descent?**
  - batch gradient descent taken to the extreme, whereby we update the weights after a single (random) datapoint is seen:

$$\nabla_{\underline{w}}[\underline{r}^T\underline{r}] \approx 2N\underline{x}^{(b)}(f^{(b)} - y^{(b)})$$

  - we can ignore the $2N$ factor, and just include it within the η for convenience

2 Classification by Bayesian Models

2.1 Multivariate Gaussian

- **How can we use multivariate Gaussians as classifiers?**
  - during training, given a set of data, for each class k, compute the **mean** $\underline{\mu}_k$ and **covariance** Σ_k
  - then, we can say that the elements of k were sampled from a **multivariate Gaussian**:

$$P(\underline{x} \mid y = k) = \mathcal{N}(\underline{x}; \underline{\mu}_k, \Sigma_k)$$

  - to **predict** a class for a new data point $\underline{x}$, we can use **Bayes' Rule**:

$$P(y = k \mid \underline{x}) \propto p(\underline{x} \mid y = k) \times P(y = k) = \mathcal{N}(\underline{x}; \underline{\mu}_k, \Sigma_k)\pi_k$$

    (to give a probability, we can just normalise using:

$$P(y = k \mid \underline{x}) = \frac{\mathcal{N}(\underline{x}; \underline{\mu}_k, \Sigma_k)\pi_k}{\sum_{j=1}^{n} \mathcal{N}(\underline{x}; \underline{\mu}_j, \Sigma_j)\pi_j}$$

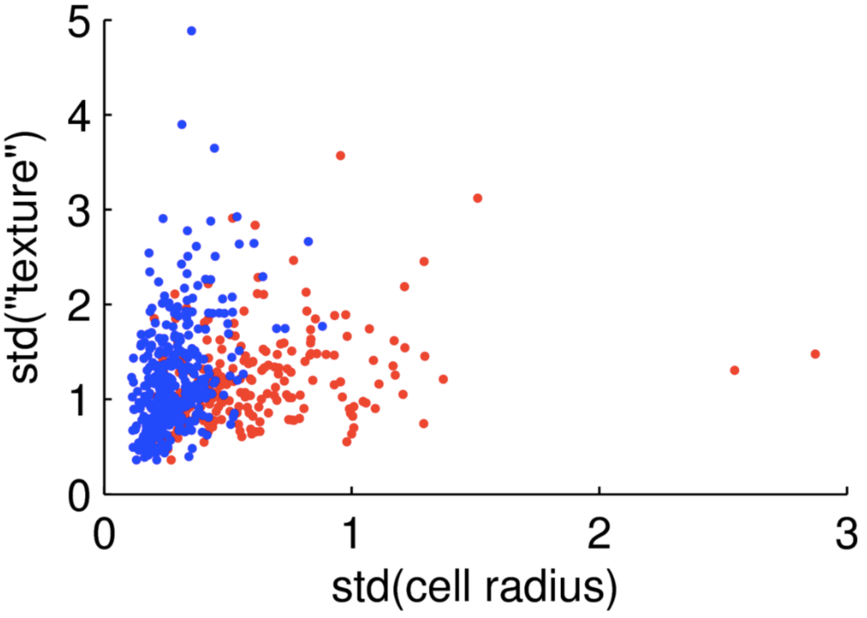
    that is, we select the class k whose Gaussian is more likely to have generated our observed data point $\underline{x}$)
  - to compute the **prior** we can just use class counts:

$$P(y = k) = \pi_k \approx \frac{1}{N} \sum_{n=1}^{N} \mathcal{X}(y^{(n)=k})$$

    where $\mathcal{X}$ is the characteristic function.

2.2 Example: Gaussians for Cancer

We have the following data:



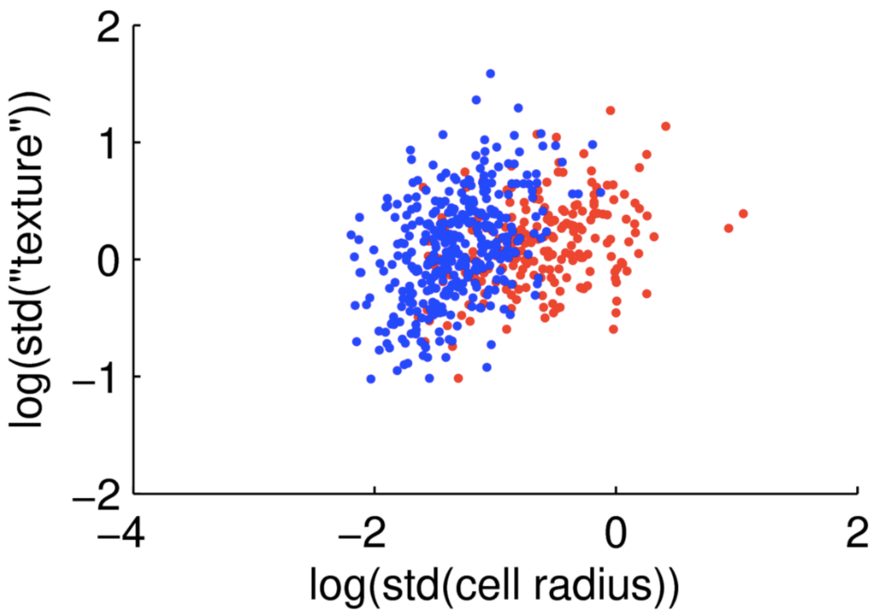
This is taken from 569 cells. Red cells are **tumorous**, whilst blue cells are healthy. The data contains more than 2 (hand-held) features, but we can't plot all of them.

> *Notice, the axes use* ***standard deviation****. This is because in each cell sample, they measured different things (i.e cell radius), but there were many cells. By using the* ***standard deviation****, they can combine all the cell information into a single number.*

Why would fitting Gaussians work badly for this example?

1. **Data Overlap**: both classes seem to be concentrated together; distinguishing between the 2 would be hard.
2. **Sign of Data**: notice, since the features are computed from standard deviation, it will be strictly positive; however, a Gaussian will assign probability density to negative values aswell.
3. **Data Isn't Gaussian**: the data looks more cone-shaped, concentrated towards the origin, and then opening up towards the sides; a Gaussian distribution doensn't fit this pattern.

However, if we take the **natural logarithm** of the data, we get:



This addresses most of the issues raised above (except possibly the data separation); however, if we look at the full-dimensional data, maybe a clear separation arises.

> *Whenever we have strictly positive data, it is a good idea to take* ***logs****. A distribution which is Gaussian on a log scale is a* ***log-normal*** *distribution, which is a rather frequent distribution.*

Care is necessary when training classifiers, especially based on the application and the type of training data. When medicine is involved, we need to ensure that the data that we have is valid. In the above example, there is a 50/50 split of cancer/non-cancer, meaning that any classifier trained on this data might assign high probabilities for cancer for most samples; however, this isn't desired behaviour: it brings about unnecessary worry. Hence, care needs to be taken when selecting the patienets used for the training set, how they were assigned to each class, and how these choices reflect on testing.

2.3 Naive Bayes

- **What is a Naive-Bayes classifier?**
  - a **classifier**, which assumes that **features** are **independent**, given the **class**
  - for example, if an input is given by $[red, 2, 1.8, Germany]$, it assumes that the **class** of this input generated each of the features individually, and that the features themselves are uncorrelated
  - this **naive** assumption allows us to write:

$$P(\underline{x} \mid y = k, \theta) = \prod_{d=1}^{D} P(x_d \mid y = k, \theta)$$

  - the probability distribution of each feature need not be Gaussian, which makes Naive Bayes a convenient classifier (since there aren't many probabilistic classifiers which work on input vectors $\underline{x}$ like Gaussians do)
- **When is a Gaussian classifier a Naive Bayes Classifier?**
  - if we use a Gaussian to model each feature, and we use $\Sigma = \mathbb{I}$ as the **covariance**, then by definition the features will be uncorrelated, and the naive assumption kicks in
- **Why would one-hot encoding break the Naive Bayes assumption?**
  - because the features will no longer be independent
  - one of the features in the input vector will be a 1, which determines that the rest of the features will be 0
- **How good is Naive Bayes?**
  - Naive Bayes tends to be either **overconfident** or **underconfident**
  - this is because the independence assumption is very strong
  - for instance, for text classification, it'll typically assing probabilities that are either 99.99% or close to 0
  - however, because of this, Naive Bayes (and Gaussian Classifiers) are good as **baselines**

3 Classification by Logistic Regression

- **What is logistic regression?**
  - a **binary classifier** which uses the **sigmoid**:

$$f(\underline{x}; \underline{w}) = \sigma(\underline{w}^T \underline{x}) = \frac{1}{1 + e^{-\underline{w}^T \underline{x}}}$$

  - recall, the **range** of σ is $(0, 1)$, so its output will be bounded within this interval
  - given some threshold p, we assign label 1 if $f(\underline{x}; \underline{w}) > p$, and 0 otherwise

- **Why do we use the sigmoid function?**
  - the fact that σ bounds $\underline{w}^T \underline{x}$ between 0 and 1 means that we can interpret $f(\underline{x}; \underline{w})$ as a **probability**
  - recall, to fit binary labels, the optimal function f was one such that:

$$f(\underline{x}) = P(y = 1 \mid \underline{x})$$

- **How does logistic regression relate to linear regression?**
  - they are both **linear**: the output depends on a linear, weighted sum of the inputs (even if σ is non-linear)
  - they can both use **basis functions**; for the sigmoid, we might as well use:

$$\sigma(\underline{w}^T \phi(\underline{x})$$

  - however, we can't use routines like `np.linalg.lstsq` to find $\underline{w}$, since the weights appear within a non-linear function

- **What loss function can be used to optimise the model?**
  - **least squares** can be used
  - however, if we interpret $f(\underline{x}; \underline{w})$ it is more natural to use **maximum likelihood**:

$$L(\underline{w}) = \prod_{n=1}^{N} P(y^{(n)} \mid \underline{x}^{(n)}, \underline{w}) = \prod_{n=1}^{N} f(\underline{x}; \underline{w})$$

    whereby we seek to maximise the probability of our model outputting the correct label

- **What is negative log-likelihood?**
  - an equivalent loss function to **maximum likelihood**, which is more computationally convenient:

$$NLL = -\log L(\underline{w}) = -\sum_{n=1}^{N} \log \left[\sigma(\underline{w}^T \underline{x}^{(n)})^{y^{(n)}} (1 - \sigma(\underline{w}^T \underline{x}^{(n)}))^{1-y^{(n)}}\right]]$$

  - we seek to **minimise** the NLL
  - here we use the "trick" that if $y^{(n)} = 1$, then the term in the summation is:

$$\sigma(\underline{w}^T \underline{x}^{(n)}) = P(1 \mid \underline{x}^{(n)}, \underline{w})$$

    whilst if $y^{(n)} = 0$ the summand becomes:

$$1 - \sigma(\underline{w}^T \underline{x}^{(n)}) = P(0 \mid \underline{x}^{(n)}, \underline{w})$$

    as expected
  - we can compact this even more by using labels in $z^{(n)} \in \{+1, -1\}$ and defining:

$$z^{(n)} = 2y^{(n)} - 1$$

    Alongside the fact that:

$$\sigma(-a) = 1 - \sigma(a)$$

    it follows that:

$$NLL = -\sum_{n=1}^{N} \log \sigma(z^{(n)} \underline{w}^T \underline{x}^{(n)})$$

  - we can also add **regularisation** to NLL

- **How does maximum likelihood compare to least squares?**
  - NLL is (asymptotically, and given a lot of data) the **most efficient** estimator
  - however, the presence of ourliers can make it so it gets extremised (since the model will be very confidently wrong)

- **Is there a closed-form solution to the negative log-likelihood?**
  - we can't analytically find $\underline{w}$ minimising NLL
  - iterative gradient methods must e used instead

- **What is the gradient vector for the negative log-likelihood?**
  - we begin by computing:

$$\begin{aligned}\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) \\ &= -\frac{1}{(1+e^{-x})^2} \times -e^{-x} \\ &= \sigma(x)\frac{1-1+e^{-x}}{1+e^{-x}} \\ &= \sigma(x)(1-\sigma(x))\end{aligned}$$

  - then, by the chain rule:

$$\begin{aligned}\nabla_{\underline{w}} NLL &= -\sum_{n=1}^{N} \nabla_{\underline{w}}(\log\sigma(z^{(n)}\underline{w}^T\underline{x}^{(n)})) \\ &= -\sum_{n=1}^{N} \frac{1}{\sigma(z^{(n)}\underline{w}^T\underline{x}^{(n)})}\nabla_{\underline{w}}(\sigma(z^{(n)}\underline{w}^T\underline{x}^{(n)})) \\ &= -\sum_{n=1}^{N} \frac{1}{\sigma(z^{(n)}\underline{w}^T\underline{x}^{(n)})}\sigma(z^{(n)}\underline{w}^T\underline{x}^{(n)})(1-\sigma(z^{(n)}\underline{w}^T\underline{x}^{(n)}))\nabla_{\underline{w}}(z^{(n)}\underline{w}^T\underline{x}^{(n)}) \\ &= -\sum_{n=1}^{N}(1-\sigma(z^{(n)}\underline{w}^T\underline{x}^{(n)}))z^{(n)}\underline{x}^{(n)}\end{aligned}$$

  - the more confident the classifier is, the larger the value of $\sigma(z^{(n)}\underline{w}^T\underline{x}^{(n)})$, so the less that the weights will get changed
  - to verify the gradient, we could use numerical approximations:

$$\frac{\partial f(w)}{\partial w} \approx \frac{f(w+\varepsilon/2) - f(w-\varepsilon/2)}{\varepsilon}$$

4 Questions

4.1 Notes Questions

1. **When are solutions to the normal equation unique? In particular, what happens if we try fitting a line through only one datapoint, a plane through only 2 datapoints, or in general fit linear regression with $D > N$?**

Intuitively, if $D > N$, we have more features than training data. For example, in $\mathbb{R}^2$, we could try fitting a line to a single point, but there are infinitely many lines. Similarly, in $\mathbb{R}^3$, there are infinitely many planes going through any 2 points. In fact, if the points are **linearly dependent**, there are still infinitely many planes going through them (for instance, think of 3 colinear points). Hence, a **unique** solution can only be derived when $N \geq D$, and there are at least D linearly independent training points.

More rigorously, say $X \in \mathbb{R}^{N \times D}$. Then:

$$rank(X) \leq N$$

Then,

$$rank(X^T X) \leq min(rank(X), rank(X^T)) \leq N$$

But $X^T X \in \mathbb{R}^{D \times D}$, so $X^T X$ won't be full-rank (as its rank is at most N), and so, it won't be invertible, meaning that the normal equation won't have a unique solution.

2. **Would L^2 regularisation solve the uniqueness problem when $D > N$?**

When optimising with L^2 regularisation, we add D new "fake" data points to the design matrix:

$$X = \begin{pmatrix} \Phi \\ \sqrt{\lambda}\mathbb{I}_D \end{pmatrix}$$

Since $\sqrt{\lambda}\mathbb{I}_D$ is full-rank, we guarantee that X will have at least D linearly independent rows/columns, and so, $X^T X$ will be invertible, and thus, a unique solution will exist.

5 Tutorial

1. **Maximum likelihood logistic regression maximises the log probability of the labels:**

$$\sum_n \log P(y^{(n)} \mid \underline{x}^{(n)}, \underline{w})$$

with respect to the weights $\underline{w}$.
Training data is said to be *linearly separable* if the 2 classes can be completely separated by a hyperplane. That means we can find a decision boundary:

$$\sigma(\underline{w}^T \underline{x}^{(n)} + b) = 0.5$$

such that all the $y = 1$ labels are on one side (i.e probability greater than 0.5), and all of the $y \neq 1$ labels are on the other side.

(a) **Show that if the training data is linearly separable with a decision hyperplane specified by $\underline{w}$ and b, the data is also separable with the boundary given by $\underline{\tilde{w}}$ and $\tilde{b}$, where:**

$$\forall c > 0, \quad \underline{\tilde{w}} = c\underline{w} \quad \tilde{b} = cb$$

The decision boundary of the sigmoid is defined by $\underline{x}$ such that:

$$\underline{w}^T \underline{x} + b = 0$$

If we multiply through by c:

$$(c\underline{w})^T \underline{x} + (cb) = 0 \implies \underline{\tilde{w}}^T \underline{x} + \tilde{b} = 0$$

Thus, the new model has the exact same decision boundary (we require that $c > 0$ so that decisions remain the same, that is, so that the side of the boundary corresponding to class 1 doesn't change)

(b) **What consequence does the above result have for maximum likelihood training of logistic regression for linearly separable data?**

- for linearly separable data, there are infinitely many possible decision boundaries (parametrised by c)
- since we seek to maximise the log-likelihood, we will seek models which are extremely confident in their predictions
- hence, we will seek weights with $c \to \infty$, which leads to models which are perfectly confident, even for new data