COMP30027 Machine Learning
Linear Regression

Semester 1, 2019

Afshin Rahimi & Jeremy Nicholson & Tim Baldwin
& Karin Verspoor

THE UNIVERSITY OF MELBOURNE

Lecture Outline

1. Hyper-Parameter Optimisation
2. Regression
3. Gradient Descent
4. Summary

Machine Learning, revisited I

How to do (supervised) Machine Learning:

0. Get hired!
1. Pick a feature representation
2. Compile data
3. Pick a (**suitable**) model e.g. Naive Bayes or K-nearest Neighbours
4. Train the model
5. Classify development data, evaluate results
6. Probably: *go to (1)*

Machine Learning, revisited II

Our job as Machine Learning experts:

- Choose a model suitable for the problem we are trying to solve.
- Choose attributes suitable for the problem and the chosen model.
- ... But what about the **hyper-parameters** of the model (e.g. K in nearest neighbour classifier)?

Hyper-parameter Optimisation

- Given an evaluation metric $\mathcal{D}$ (like Accuracy), a dataset $\mathcal{T}$, a feature representation $\mathcal{F}(\mathcal{T})$, and a learner $\mathcal{L}$ with hyperparameters θ^h:
- Maximise $\mathcal{D}(\mathcal{L}, \theta^h; \mathcal{F}(\mathcal{T}))$
- (More commonly, minimise a "loss" metric, like Error Rate)
- Holding $\mathcal{F}(\mathcal{T})$ and $\mathcal{L}$, fixed:

$$\hat{\theta}^h = \arg\min_{\theta^h \in \Theta} \mathrm{Error}(\mathcal{L}, \theta^h; \mathcal{F}(\mathcal{T}))$$

Grid Search

$$\hat{\theta}^h = \arg\min_{\theta^h \in \Theta} \mathrm{Error}(\theta^h; \mathcal{L}, \mathcal{F}(\mathcal{T}))$$

- Analytic solution (i.e., when closed form can be computed exactly):
  - requires solving $\frac{\partial(\mathrm{Error})}{\partial\theta_1} = \ldots = \frac{\partial(\mathrm{Error})}{\partial\theta_D} = 0$
  - derivatives are not calculable (defined?) in this context, so we can't use an analytic solution.

Iterative Grid-search for KNN

```
best_score = 0
best_params = None for k in [1, 2, 3]:
   for metric in ['euclidean', 'cosine']:
      knn = KNeighborsClassifier(k, metric):
      knn.train(X_train, y_train)
      score = knn.score(X_dev, y_dev)
      if score > best_score: best_score=score, best_params =
(k, metric)
```

Train a new model on training data with best_params and evaluate it on test data.

Grid Search or Exhaustive Search

- As we saw in KNN, we exhaustively search the parameter space.
- For each **numerical** $\theta_k \in \mathbb{R}$:
    - Identify boundaries of range R_-, R_+
    - Divide range $[R_-, R_+]$ into linear or logarithmic steps.

Hyper-parameter Tuning

- Usually used as a final stage, to get higher Accuracy with respect to the development data
- Because we are evaluating lots of models, there is a risk of "over-tuning":
  - the best choice of hyper-parameters for the development data may not the best choice of hyper-parameters on the test data
  - (special case of "over-fitting", more in Evaluation II)
- If you are comparing two different models, you should do the same number of hyper-parameter search iterations for both.
- There are other methods for hyper-parameter tuning other than Grid Search: Random Search, Evolutionary, Gradient-based, and Bayesian optimisation.

Lecture Outline

Categorical Features and Classes

- We have studied Naive Bayes and Decision Tree classifiers:
  - The training data consists of input-output pairs.
  - Input: categorical or numeric, Output: categorical (e.g. Yes/No, dog/cat, spam/notspam).
- What if the output/class is **continuous**?

Regression

- Important type of Machine Learning problem where the output is continuous (e.g. is a real number) with many applications:
  - Predict wind farm energy output from weather data (we need robust energy sources, so we need to be able to predict the output)
  - Predict the number of customers for a shop from date/weather/holidays/ (we need to know how many sales personnel we need in each day rather than rely on intuition)
  - Predict life expectancy of critical patients (to create the best treatment plan for them).
  - Predict the price of a product (e.g. gold/stocks) in future (for economic planning).

Linear Regression

- Continuous attributes $\rightarrow$ continuous class
- Assuming a linear relationship between the k attribute values a_i and the numeric output c:

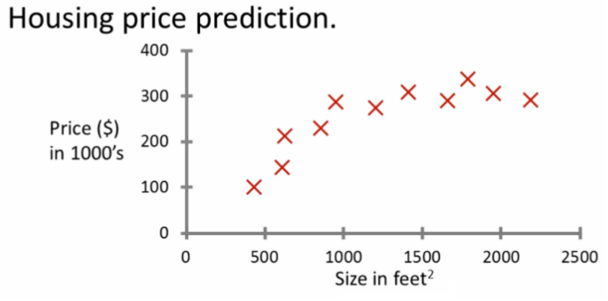
$$c = w_0 + \sum_{i=1}^{k} w_i a_i$$

where w_i is a weight corresponding to a_i

Example: Supervised Learning (Regression)

Can we predict housing prices?

A friend has a house which is 750 square feet.
How much can he expect to get?



Linear regression, mathematically

Linear regression captures a relationship between two variables or attributes.

It makes the assumption that there is a *linear* relationship between the two variables.

- An outcome variable (aka response variable, dependent variable, or label)
- A predictor (aka independent variable, explanatory variable, or feature)

At its most basic, the relationship can be expressed as a *line*:

$$
\begin{aligned}
y &= f(x) \\
y &= \beta_0 + \beta_1 x_1 + \ldots + \beta_D x_D \\
y &= \beta \cdot x \text{ (given } x_0 = 1\text{)}
\end{aligned}
$$

A simple assumption!

Linear functions are less descriptive than non-linear functions, but permit simpler (mathematical) strategies.

They capture changes in one variable that correlate linearly with changes in another variable.

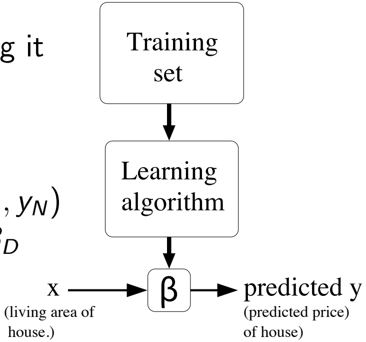
For some variables, this makes sense. For example: The more umbrellas you sell, the more money you make. How much money you make is directly proportional to how many umbrellas you sell.

Training & Prediction

We derive a linear model by estimating it from training examples.

Given examples $(x_1, y_1), (x_2, y_2), ...(x_N, y_N)$, we determine the optimal $\beta_0, \beta_1, ...\beta_D$

Armed with a linear model $y = \beta \cdot x$, we can straightforwardly predict a continuous valued output for $\hat{y}$ given a value of x.



Fitting the model I

We want to choose the *best* line.

- Operationally, the line that minimises the *distance* between all points and the line.
  - Recall Euclidean distance: $d(A, B) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$
- *Least squares estimation*: find the line that minimises the sum of the squares of the vertical distances between approximated/predicted $\hat{y}_i$s and actual y_is.
  - **Minimise** the Residual Sum of Squares (RSS) (aka Sum of Squares Due to Error (SSE)):

$$\begin{aligned} RSS(\beta) &= \sum_i (y_i - \hat{y}_i)^2 \\ &= \sum_i (y_i - \beta \cdot x_i)^2 \end{aligned}$$

Fitting the model II

$$\begin{aligned} \hat{\theta} &= \operatorname*{arg\,min}_{\theta \in \Theta} \mathrm{Error}(\theta; \mathcal{L}, \mathcal{F}(\mathcal{T})) \\ \hat{\beta} &= \operatorname*{arg\,min} \mathrm{RSS}(\beta; \{\boldsymbol{X}, Y\}) \end{aligned}$$

- Just a special case of the optimisation problem from before
- All attributes are numerical $\rightarrow$ Grid Search is)-:
- Partial derivatives can be (easily!) calculated
- (RSS is **convex** — the local optimum is a global minimum)

Fitting the model III

Derivatives of RSS, with respect to weight vector β, for N instances, and D attributes:

$$\frac{\partial}{\partial \beta_0} = -2 \sum_{i=1}^{N} (y_i - \hat{y}_i)$$

$$\frac{\partial}{\partial \beta_k} = -2 \sum_{i=1}^{N} x_{ik} (y_i - \hat{y}_i)$$

- We could set everything to 0, and then solve $D + 1$ equations with $D + 1$ unknowns
- ... But the matrix problem is subject to numerical errors...

Lecture Outline

From earlier...

Iterative approximation to Error optimisation:

1. **Initialisation:** Guess θ^0, set $i = 0$
2. **Evaluate:** Compute $\mathrm{Error}(\theta^i; \mathcal{L}, \mathcal{F}(\mathcal{T}))$
3. **Termination:** Decide whether to stop (break if so)
4. **Update:** Set θ^{i+1} from θ^i somehow; $i = i + 1$
5. Go to step 2

More now!

Gradient Descent Algorithm I

$$\begin{aligned}
\theta^{i+1} &:= \theta^{i} - \alpha \nabla \mathrm{Error}(\theta^{i}) \\
\theta_k^{i+1} &:= \theta_k^{i} - \alpha \frac{\partial}{\partial \theta_k^{i}} \mathrm{Error}(\theta^{i}) \\
\beta^{i+1} &:= \beta^{i} - \alpha \nabla \mathrm{RSS}(\beta^{i}; \boldsymbol{X}, Y)
\end{aligned}$$

Substituting the partial derivatives gives us:

$$\beta_k^{i+1} := \beta_k^{i} + 2\alpha \sum_{j=1}^{N} x_{jk}(y_j - \hat{y}_j^{i})$$

Gradient Descent Algorithm II

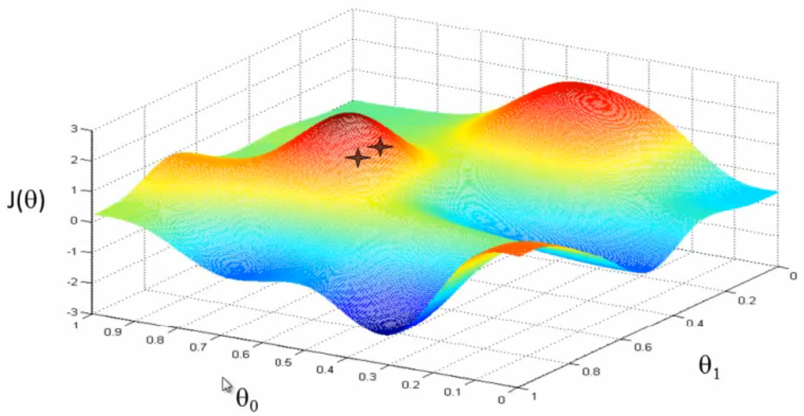
- Steps in the Gradient Descent algorithm involve:
  - making a prediction for each (training) instance
  - comparing the prediction with the actual value
  - multiplying by the corresponding attribute value
  - updating the weights after all of the training instances have been processed
- We were going to compare the predictions with the actual values anyway, when we evaluate the model!
- (Note that the evaluation metric is now included in the model...)

Gradient Descent Algorithm III

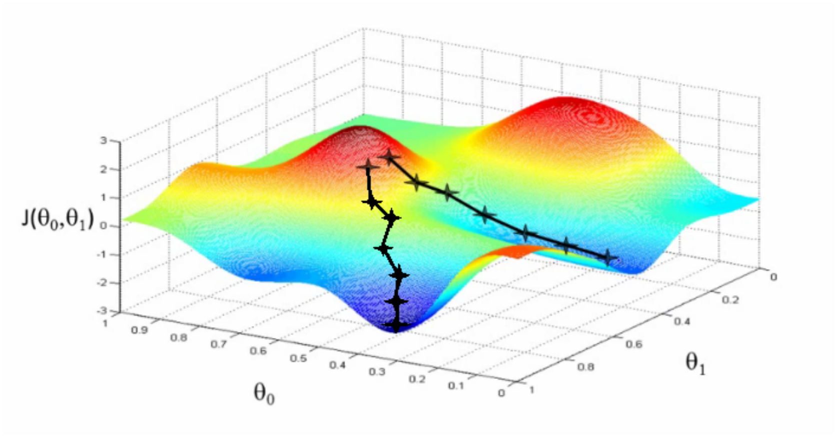
Logic:

- $\nabla \mathrm{Error}(\theta)$ is a vector of partial derivative terms. It measures the slope (= gradient) of the function $\mathrm{Error}(\theta)$: this is the direction
- The gradient points up-hill; we follow it down-hill. Each update reduces the error slightly.
- α is a parameter of the algorithm, representing the *learning rate* (how big a step you take in updating θ_i).
- If α is too small, the algorithm might be slow. If it is too large, you might miss the minimum.

Gradient Descent pictorially



Gradient Descent pictorially



Linear Regression over Larger Feature Sets

- CPU dataset:

$$\begin{aligned} PRP &= -56.1 + 0.049 MYCT + 0.015 MMIN + 0.006 MMAX \\ &\quad + 0.630 CACH - 0.270 CHMIN + 1.460 CHMAX \end{aligned}$$

MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP
125	256	6000	256	16	128	199
29	8000	32000	32	8	32	253
29	8000	32000	32	8	32	253
29	8000	32000	32	8	32	253
29	8000	16000	32	8	16	132
26	8000	32000	64	8	32	290
23	16000	32000	64	16	32	381

$\vdots$

What about Nominal Attributes?

- We can easily map nominal attributes onto numeric attributes through binarisation
- If we treat each resulting binary feature as continuous, we can use linear regression as is

Evaluation of Numeric Prediction

- It clearly doesn't make sense to evaluate numeric prediction tasks in the same manner as classification tasks, as:
  - "direct hits" (true positive matches) are an unreasonable expectation
  - unlike classification, we can make use of the inherent "ordering" and "scale" of the outputs
- There are many, many scoring metrics for regression tasks, all of which are based on the absolute or relative difference between the predicted ($\hat{y}_i$) and actual (y_i) values of test instances

Evaluation metrics

- In addition to RSS, which is already present in the model:
- Mean squared error (MSE):

$$\frac{1}{N}\sum_{i}(\hat{y}_i - y_i)^2$$

- Root mean-squared error (RMSE):

$$\sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}}$$

- Root relative squared error: (relative to baseline) (RRSE)

$$\sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}}, \text{ where } \bar{y} = \frac{\sum_i y_i}{N}$$

Evaluation metrics (more)

- Correlation coefficient (Pearson's correlation):

$$r = \frac{S_{\hat{Y}Y}}{\sqrt{S_{\hat{Y}} S_Y}} \quad \text{where} \quad S_{\hat{Y}Y} = \frac{\sum_i (\hat{y}_i - \bar{\hat{y}})(y_i - \bar{y})}{N-1}$$

$$S_{\hat{Y}} = \frac{\sum_i (\hat{y}_i - \bar{\hat{y}})^2}{N-1}$$

$$S_Y = \frac{\sum_i (y_i - \bar{y})^2}{N-1}$$

(statistical correlation between predicted and actual values)

Which Metric to Use?

- The relative ranking of methods each across the different metrics is reasonably stable, such that the actual choice of metric isn't crucial

	A	B	C	D
Root mean-squared error	67.8	91.7	63.3	57.4
Root relative squared error	42.2	57.2	39.4	35.8
Correlation coefficient	0.88	0.88	0.89	0.91

Beyond Linear Methods

- Linear regression is an intuitive, (relatively) easily implementable method, but can we usually expect a linear relationship between feature values and target variables?
- Features can be mapped to a higher dimension where they are linearly separable: e.g. $(x_1, x_2) => (x_1, x_2, x_1^2, 2x_1x_2, x_2^2)$
- Non-linear methods for numeric prediction include:
  - regression trees
  - neural networks
  - support vector regression

Lecture Outline

Summary

- How hyper-parameters are chosen?
- What is linear regression, and how does it operate?
- What is gradient descent, and why is it used for linear regression?
- How can we evaluate regression tasks?