DataSci 420

lesson 4: feature selection

Seth Mottaghinejad

today's agenda

- curse of dimensionality
- how the curse of dimensionality relates to overfitting
- why do feature selection?
- how to do feature selection:
    - filter methods
    - wrapper methods
    - embedded methods
- what is regularization

curse of dimensionality

- two points apart by 1 unit in each dimension have a distance of 1 in 1D, $\sqrt{2}$ in 2D, $\sqrt{3}$ in 3D and so on
- if you have more features (columns), you need to compensate it with more data (rows) otherwise you have **data sparsity**
- closely related to **overfitting**, because more features
  - **increases model complexity** $\rightarrow$ more likely to overfit
  - **increases data sparsity** making it harder to generalize $\rightarrow$ more likely to overfit

feature engineering vs feature selection

- if **feature engineering** is about adding features, **feature selection** is about subtracting features
- *"perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away"* Antoine de Saint-Exupery
- but why bother with feature selection? isn't more "information" always better? no, what's good is
  - **more relevant** features: relevant to predicting the target
  - **less redundant** features: low colinearity between features
  - **more examples** (rows) to compensate for having many features

how to do feature selection

- **filter methods:** do this **before** training by removing features that share high **correlation** or **mutual information**
- **wrapper methods:** train **many times** each time using a different subset of features
  - example: **step-wise regression** aka **best subset regression**
- **embedded methods:** build feature selection into the algorithm itself, also called **regularization**
  - example: **LASSO regression**

break time

pros and cons

- **filter methods** are time-consuming and can be a little subjective (e.g. which feature should I drop, what correlation threshould should I pick)
- **wrapper methods** are inefficient and can have **high variance** (same results can be hard to replicate), but they are easy to interpret and require little intervention
- **embedded methods** offer the best of both worlds: they are efficent and built into the algorithm itself, and through the **regularization constant** they are **tunable**

regularization

- **reduces overfitting** and the extent of it can be adjusted or **tuned**
- instead of minimizing prediction error only, minimize **prediction error** L **+ a penalty term** R where the penality term is higher when model coefficents are higher

$$\text{minimize}\ \{L(\text{actual} - \text{predicted}) + \lambda R(\text{model})\}$$

- prefers models with **smaller coefficents** (this only makes sense if you normalize features first)
- can in some cases result in **feature selection** as a by-product

the end