Supervised Data Sets

Week	TUES. Week 3
Assignment Due	
Assignment Done	□
Due Date	
Notes Done	☑

Presentation

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/f2a9e04e-ae9f-4c66-b78c-292386e36fce/L1-SupervisedDataSets.pdf

Class Notes

Supervised Datasets

Supervised and unsupervised data have ***observable data***, X.

- However, ***supervised data*** also comes with the ***target values*** as Y.
- PCA was an example of an ***unsupervised algorithm***, as it didn't need the target values to do its job.
- Conversely, feature selection via entropy, was an example of a ***supervised approach***, since we needed to know the target class labels to compute the entropy.

Evaluating Supervised Datasets

If we have target values, *evaluating* the quality of a machine learning is relatively easy.

- Let observation x have target value y.

    - Then, a given ML algo. can make a *prediction* for this observation as y'.
    - Depending on what our task is, we can use y and y' to compute different scores.
- If we have supervised data, we can evaluate how well we're doing by several ways, which vary in value by the task that we're doing.

In supervised learning, we have two problems we'll tackle:

- **Regression** → Objective: Trying to compute a continuous, unbounded value
    - In this problem, we can use the **squared error**. Since we aren't only focused on one observation, we can do this over the entire dataset.
        - Single observation: $SE = (y - \hat{y})^2$
        - Entire dataset (X, Y): $SE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2$
    - When talking about error conceptually, we're used to talking about the root of the squared error.
        - Numerically, it's more natural to think about the error, *not* the squared error, giving us the root of the squared error, known as the **root mean squared error (RMSE):**
            - $RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2}$
            - This gives us an quantitive measure of our error
        - Issue with RSME and squared error: These measures are ***scale-dependent***.
            - If target values are between 0 and 5,000, the squared error and RSME will be affected.
            - To solve this, we can think about how close we are in a *relative* way.
                - Instead, we can use the **symmetric mean absolute percent error (SMAPE)**, defined as:

$$SMAPE = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|Y_i - \widehat{Y_i}\right|}{|Y_i| + \left|\widehat{Y_i}\right|}$$

        - It is the average of all observations, looking at absolute value of difference, and sum of absolute values.
        - This now gives us a percentage between 0 and 1 that is more interpretable.

- **Classification** → trying to produce an enumerate class ID
    - In classification, using RSME or SMAPE doesn't make sense since classes that may be categorical in nature. It doesn't make sense to subtract these labels from one another.
    - Instead, we can talk about the **accuracy** of our system, calculated as the percentage of time we're correct.

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N} Y_i == \widehat{Y_i}$$

        - Each observation has an enumerated class ID, which our system outputs as well. We determine how many times we're correct and divide by the number of observations we have.

In **classification**, we have different types of classification that have specific ways that we can measure accuracy within their respective types:

- **Binary classification** - when we have only two possible classifications
    - There are many different ways that we can refer to the two classes:
        - 0 vs 1
        - 1 vs 2

    - Positive vs Negative… and many more!
  - If we refer to the two possible classes as the positive and negative class, then we have four different possibilities:
    - **True positive** = Hit
    - **True negative** = Correct rejection
    - **False positive** = False Alarm (Type 1 error)
    - **False negative** = Miss (Type 2 error)
  - From the four error types, we can establish some binary-classification-specific measurements.
    - **Precision** - % of things *classified* as positive and *actually* were positive
      - $Precision = \frac{TP}{TP+FP}$
    - **Recall (also called true positive rate, TPR)** - asks "among all real positives, what % of this did we correctly ID as positive?"; the percentage of true positives (***sensitivity***) correctly ID'd
      - $TPR = Recall = \frac{TP}{TP+FN}$
      - Takes into account true positives *and* false negatives; want to maximize true positives, while *minimizing* false negatives
      - Can also think of this as total observations we made that were positive and *total of actual positives* that exists: $\frac{TP}{P}$
    - In an ideal world, we'd want our precision and recall to be high. However, these measures often have a tradeoff in deciding on a threshold that affects the precision and recall balance. In light of this, we can think of the **f-measure**:
      - **F-measure** - the weighted harmonic mean of precision and recall
        - $F_1 = \frac{2*precision*recall}{precision+recall}$
    - **False positive rate** - calculated as $FPR = \frac{FP}{FP+TN}$
    - Example that calculates these measures: Imagine we have 100 observations → of these observations, 20 were positive and 80 were

negative

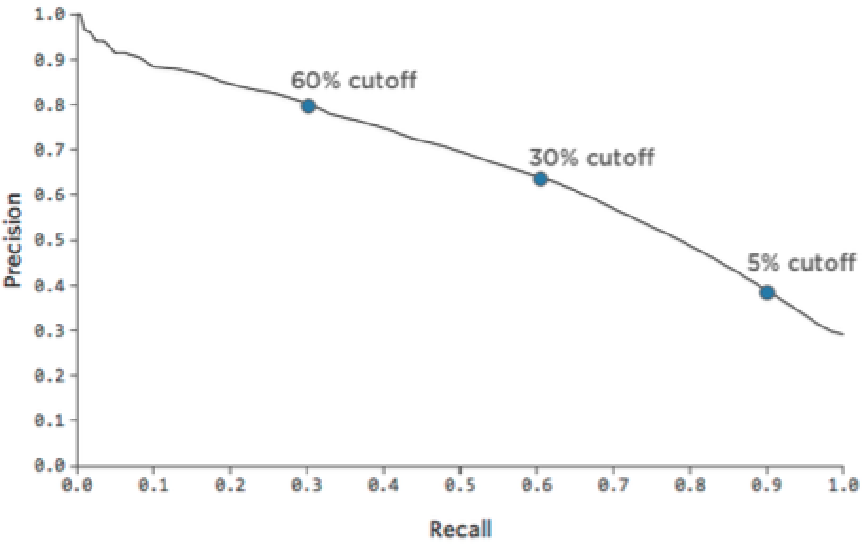
    - Our classifier classified 10 true positives, 60 true negatives, 20 false positives, and 10 false negatives
    - **Recall** → $\frac{10}{10+10} = 0.50$
    - **Precision** → $\frac{10}{10+20} = 0.333$
    - **Accuracy** → $\frac{TP+TN}{N} = \frac{70}{100} = 0.70$
    - These measurements give us a way to truly classify the performance of a classifer rather than with a simple measure with accuracy.

- Some classifiers don’t simply return what class an observation belongs to, rather returns the probability of belonging to that class → $P(y = i|x)$
  - In these cases, we can use a ***threshold*** to determine what class an observation belongs to, which logically can start at 50%.
    - For instance, with binary classification, we can say:

$$\hat{y} = \begin{cases} Positive & P(y = Positive|x) > t \\ Negative & otherwise \end{cases}$$

      - As noted above, at times there a trade off between precision and recall and adjust them accordingly.
        - We can explore the effect of this threshold on the precision and recall values.
        - The plot of precision vs. recall as a function of threshold creates a <u>**precision-recall curve (PR)**</u>.
          - <u>Ex:</u> If we set the threshold to 0%, most, if not everything, will be classified as the positive class (since it’ll only need to be above 0%).
          - Based on this, we get a lot of recall since we’ll capture every positive.

        - However, this is at a trade-off of precision since we won't actually be capturing the positive class accurately.



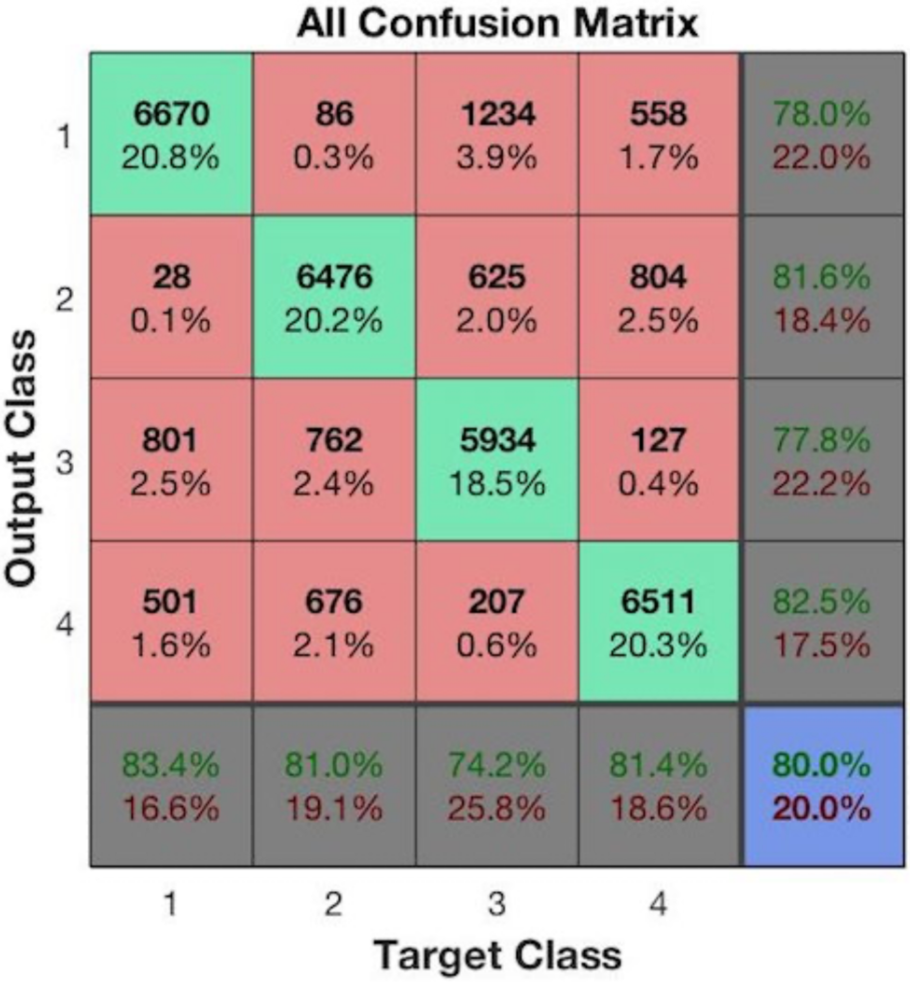
    - To evaluate a binary classifier, we can also compute the **area under the curve (AUC)** of a PR curve.
        - Given points on the curve, (R_k, P_k), we can approximate the AUC as:

$$AUC = 1 - \frac{1}{2}\sum_{k=1}^{n}(P_k + P_{k-1})(R_k - R_{k-1})$$

        - An ideal PR curve will have an AUC of 1.0.
- **Multi-class classification** - when we have several possible classifications
    - Like binary classification, we can evaluate the accuracy of a multi-class classifier:

$$accuracy = \frac{1}{N}\sum_{i=1}^{N}(Y_i = \widehat{Y_i})$$

- An additional classifcation we can add to MCC, we may be interested in investigating which classes get confused with which other classes.
    - Ex: Does it evenly distribute among other classes that it is incorrectly classifying? If we ID where the confusion is occurring, we can adjust our classifer accordingly by adjusting any hyperparametesr that we have control of.
    - To observe this, we can look at a **confusion matrix**.
        - For a small number of classes, we can create a color-coded matrix that is easy to follow.
            - The diagonal is the ideal situation since it is the rate at which classes were *accurately* classified.

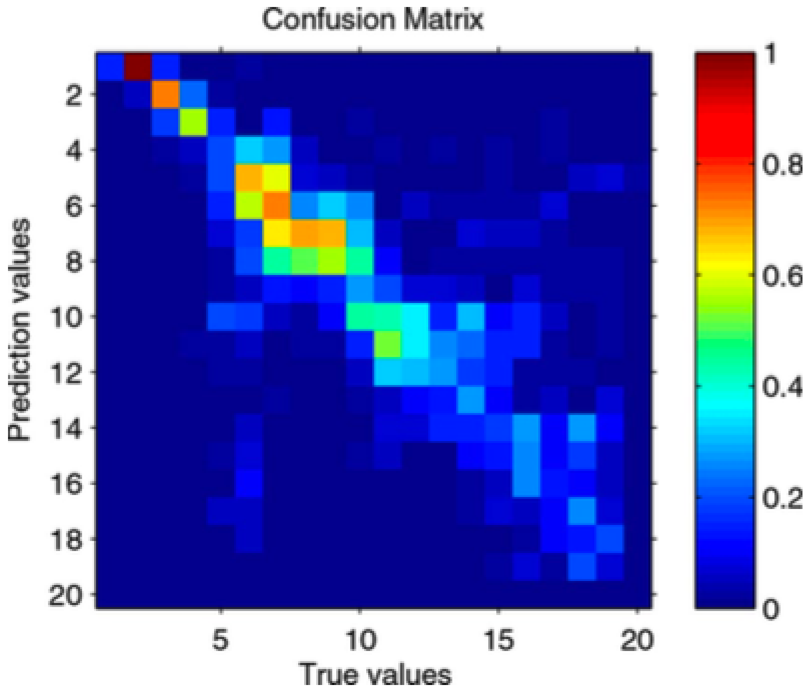


All Confusion Matrix

Output Class	1	2	3	4	
1	6670 20.8%	86 0.3%	1234 3.9%	558 1.7%	78.0% 22.0%
2	28 0.1%	6476 20.2%	625 2.0%	804 2.5%	81.6% 18.4%
3	801 2.5%	762 2.4%	5934 18.5%	127 0.4%	77.8% 22.2%
4	501 1.6%	676 2.1%	207 0.6%	6511 20.3%	82.5% 17.5%
	83.4% 16.6%	81.0% 19.1%	74.2% 25.8%	81.4% 18.6%	80.0% 20.0%

Target Class

- Ex: For class 3, it seems as if it is disproportionately assigned to class 1. In identifying this, we can begin to evaluate where the confusion may be occurring.



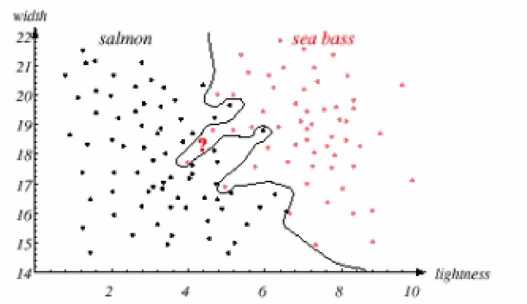
- Each type of classification can have different types of evaluations that we can do.
  - Many of the classification algorithms we'll look at are designed to work for binary classification, specifically logistic regression and support vector machines (SVMs).
    - To support multiple classes using binary classifier, we can use a "one vs one" or "one vs all" approach:

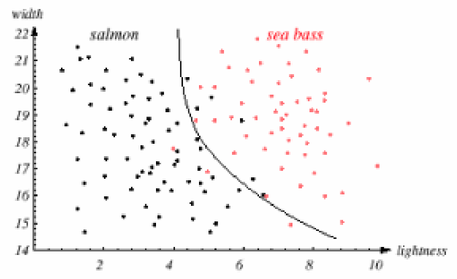
      1. **One vs all (need *K* classifiers)** - often imbalanced, ambiguous regions
         a. Choose the one that does best.
      2. **One vs one ($\frac{K(K-1)}{2}$)classifiers** - lots of classifiers
         a. Fewer ambiguous regions
         b. Choose the one that gets the most votes.

Generalization

Given enough observations, it is relatively easy to build a system that does well on the data it is trained on.

- However, this is not an interesting problem to solve.
    - The key is to be able to make predictions on data that has NOT been trained on before.
    - We ask: How well does the system we built, using training data, **generalize** on other data?
    - Ex: We have training data → a small subset of data we *could* obtain
        - In a system able to 100% classify classes on our training data, when applying it to novel data, we see that it does not work well in spite of our high performance with our training data.
            - The system likely generated an artifiically complex function for classification of the training data, not the function of the entire set of possible data.
                -





-
- With a less accurate system that does generalize, it may only get the big picture.

Dataset Splitting

To help us determine how well our model generalizes, we typically split out data into two groups:

- **Training data**
- **Validation data**

- As a starting point, we may split the data as a $\frac{2}{3}$ training and $\frac{1}{3}$ validation set.
    - We can then build/train our system using the training data and check the generalizability of our system using the validation set.
- It is important to note that more data is NOT always better.
    - The key to a good model is to have the training data and validation data pulled from the *same distribution.*

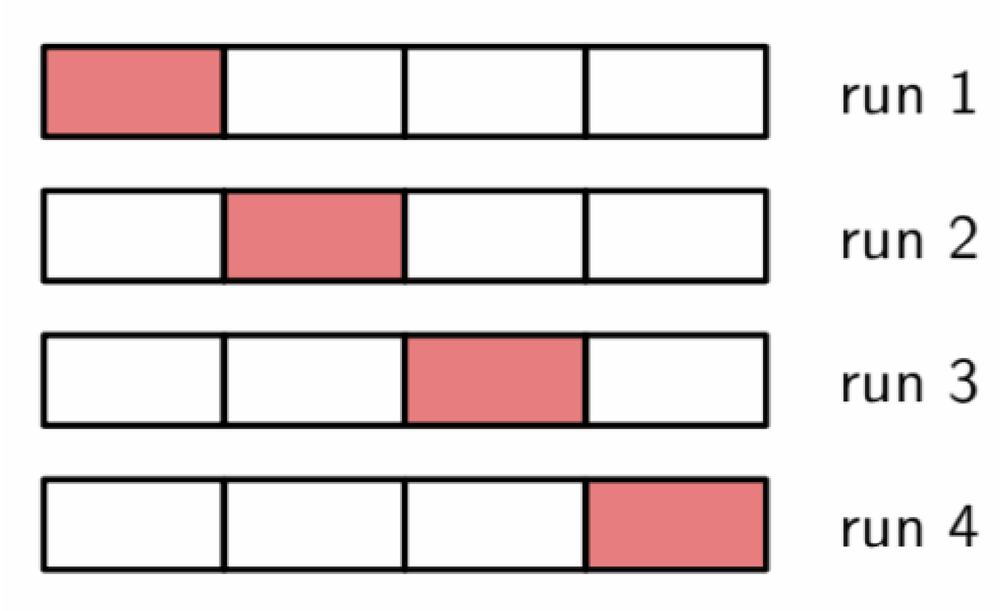
Cross Validation

If we use $\frac{2}{3}$ of our data for training and $\frac{1}{3}$ for training, if we already don't have much data, the $\frac{1}{3}$ validation data will be very small for sampling.

- A technique we can use to address this is **cross-validation**.
    - It takes more time, but allows us to use vast majority of data for training and only little for validating *multiple times*, accruing a good number of validation samples ultimately.
        - In having several training/validation runs, we keep track of all of the errors.
        - We can then compute statistics for our classifier based on the list of errors.
    - In the end, our final system will be built using *all the data*.

There are a few different types of cross validation:

- **S-folds cross validation (also referred to as K-folds cross validation)** - says to take the dataset and break it into S equal parts, train on $S-1$ of them and validate the remaining part, repeating S times
    - Ex: Have 100 observations and want to do 4-folds cross validation
        - This means each subset will have 25 samples
        - 4 times, we'll grab a chunk for validating and the remaining $S-1$ chunks will be for training



  - When all is done, we have accrued a total of 100 validation points across all systems, using 75% of the data for training each training/validation run.

- **Leave-one-out** - if our dataset is really small, we may want to build our system on $N - 1$ samples and validate on just one sample and do this N times (basically N-folds)