EECS445 W24:
Performance Metrics, Cross Validation, and Feature Selection Notes

EECS445 Staff

February 1, 2024

1 Performance Metrics

1.1 Confusion Matrix and Metrics

Often in machine learning we talk about obtaining a "better model" for classifying our data. We've talked mostly about using classification error as our metric, but there are other performance measures that we can use.
To start, let us introduce the confusion matrix.

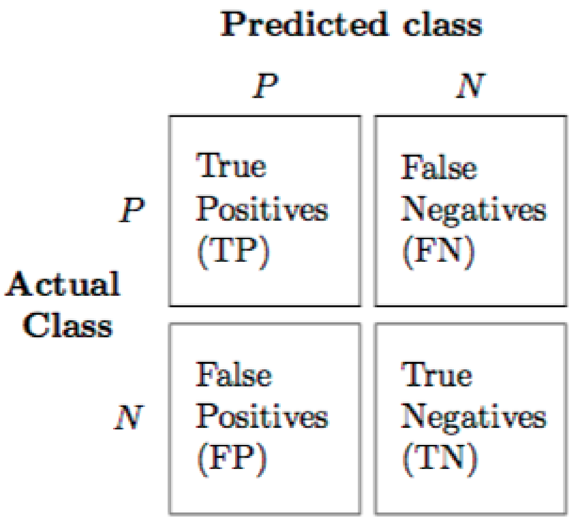

Figure 1: Confusion Matrix

1. True Positives (TP): Examples that are positive and we classified as positive
2. False Negatives (FN): Examples that are positive and we incorrectly classified as negative (our negative classification was false)
3. False Positives (FP): Examples that are negative and we incorrectly classified as positive (our positive classification was false)
4. True Negatives (TN): Examples that are negative and we classified as negative

Based on these, we can define the following performance metrics:

1. Accuracy: $\frac{TP+TN}{Total}$
2. Sensitivity/Recall/TP Rate: $\frac{TP}{TP+FN}$
3. FP Rate: $\frac{FP}{FP+TN}$
4. Specificity: $\frac{TN}{TN+FP}$
5. Precision (PPV): $\frac{TP}{TP+FP}$
6. F1 Score: $2\frac{Precision*Sensitivity}{Precision+Sensitivity}$

Discussion Question 1. *Is it possible to maximize your score in all measures? When might you want to use certain measures over others (e.g. Sensitivity vs. Specificity)?*

Solution: *It isn't possible to maximize all scores at once with an imperfect classifier or without a perfectly separable data-set. There is a trade-off amongst some of these. For example, if we wanted to maximize specificity, we could classify everything as negative and get a perfect score of 1. Conversely, if we wanted to maximize sensitivity, we could classify everything as positive and get a perfect score of 1.*

1.2 Receiver Operating Characteristic Curve (ROC)

Recall the definitions of specificity and false positive rate which we just introduced. In this section, we will show how we can use the tradeoff between them to define another useful performance metric, the ROC Curve.

- Sensitivity/Recall/TP Rate: $\frac{TP}{TP+FN}$
- FP Rate: $\frac{FP}{FP+TN}$

The Receiver Operating Characteristic (ROC) Curve is a curve that shows the tradeoff between the true positive rate and false positive rate. It indicates the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. Using the ROC curve, we obtain another performance metric for our classifiers: the Area Under the ROC Curve (AUROC). A perfect AUROC score would be of 1.00 whilst a classifier that performs no better than random would have a score of 0.50.

ROC curve can be used to select a threshold for a classifier which maximizes the true positives, while minimizing the false positives.

To obtain the ROC Curve, we would first obtain the classification score of each point (note: not only would we use the sign of the classification, but also the magnitude). Then, we arrange them in descending order. We start by classifying everything as positive and progressively move up one-by-one our classifications in ascending order until we classify all points as negative. Whenever we move our classification line up, we plot the TPR and FPR at that time.

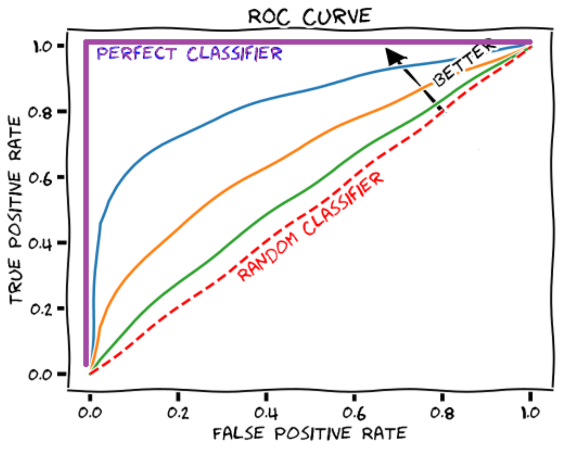

Figure 2: AUROC Curve (image from glassboxmedicine.com)

2 Cross-Validation

Often, our algorithms come with **hyperparameters**, that is, numbers that must be tuned outside of training. For instance, the strength C of regularization for soft-margin SVMs is a hyperparameter. Usually hyperparameter-search occurs by **trial and error**: we try many hyperparameter values, and for each value, we train the main algorithm (two popular implementations are **random search** or **grid search**). Then we stick with the value that yields the lowest loss.

To avoid overfitting, we must calculate **validation loss** on a different set of data than that on which we trained. Otherwise, we will commit the same mistake as the teacher who measures which teaching technique works best by making the Final Exam a duplicate of the Midterm (this just tests memorization, not general understanding). So, each step of the learning process uses a separate dataset. The most common names for these sets are:

1. Test set
2. Validation set
3. Train set

For instance, one might randomly shuffle one's datapoints, then allocate 10% of samples to the test set, 10% to the validation set, and 80% to the train set.

See Figure 3 for a depiction of these splits.

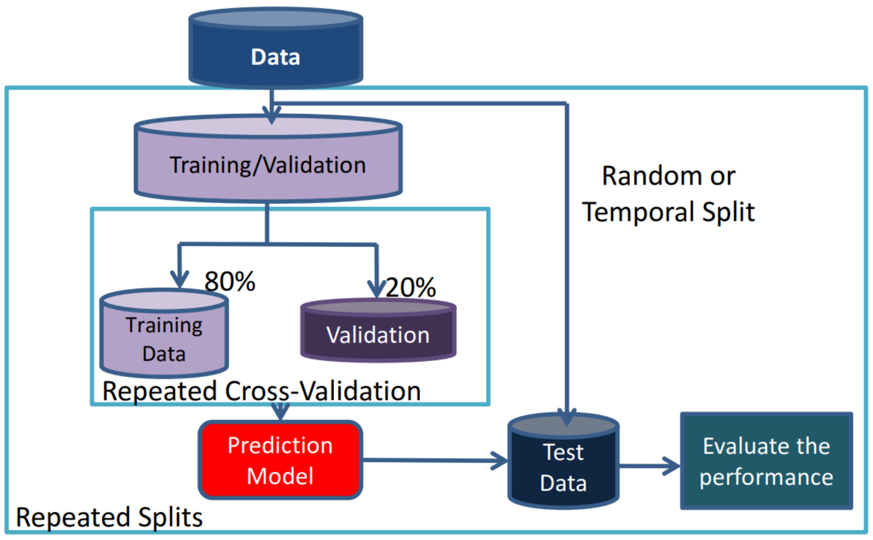

Figure 3: A flowchart depicting the creation of train, test, and validation sets from an original dataset.

However, a main drawback of just holding-out a single portion of the entire dataset for validation is that we are evaluating hyperparameters based on the performance on just one validation dataset. One way to help alleviate this problem is **k-fold cross-validation**. In this approach, we first randomly split the training data into k partitions. We pick one of the partitions to use for validation and train a model on the other $k-1$ partitions. This is repeated for each of the k partitions and the hyperparameter settings are evaluated based on the average validation performance across all k partitions. Note that this requires us to train k different models which can take quite a bit longer to complete.

3 Feature Selection

The goal of **feature selection** is to systematically decide which features are most relevant to the problem at hand. Feature selection can also be used for data visualization; for example, by plotting the two or three most salient features together or projecting a high-dimensional dataset onto a lower-dimensional space.
There are simple feature selection methods that try to discard irrelevant features, such as filter and wrapper methods; these often involve finding correlation between performance and features or subsets of features. The goal of feature selection is to systematically decide which features are most relevant to the problem at hand. In the era of “big data”, it is not uncommon for companies to work with datasets containing millions of features per datapoint. At this scale, feature selection has the potential to drastically reduce storage requirements, training time, and overall costs.

Filter vs. Wrapper Methods

The simplest feature selection methods attempt to discard irrelevant features, leaving others intact. Algorithms of this type can be divided into two categories:

- **Filter methods** rank all features from most- to least-relevant and keep only the top-n best. Features are typically ranked by their correlation or mutual information with the target value, and dependencies between features are ignored.
- **Wrapper methods** wrap the learning algorithm inside a search algorithm that searches for the best subset of features. Each subset is considered as a whole, and redundant variables are kept only if they are found to help performance.

Embedded Feature Selection

Some models come with a feature selection mechanism built-in. Typically, this is achieved by encouraging **sparsity** in the model parameters. For example, if the vector of weights learned for a linear regression model contains many zeros, it is probably safe to discard the corresponding features, as they have little or no impact on the final prediction. There are several common ways to encourage sparsity in a model. Here, we will focus on the use of **regularization** as a means of feature selection, specifically the ℓ_0 and ℓ_1 regularization terms. Recall the definition of the ℓ_p-norm.

$$\ell_p = \sqrt[p]{\sum_i |\theta_i|^p} \tag{1}$$

The ℓ_0-norm—while not strictly a norm—represents the optimal solution to our issue of sparsity by minimizing the number of non-zero elements in a vector. As this yields a binary optimization (i.e., NP-Hard) problem, we tend not to use it. Instead, we often use the ℓ_1-norm to encourage sparsity.

Discussion Question 2. *How does the ℓ_1-norm encourage sparsity? (Hint: consider the gradient functions in Figure 4 and Figure 5.)*

Solution: *Consider the gradient function of the ℓ_1- and ℓ_2-norms (see Figure 4 and Figure 5). Note that as the magnitude of our vector approaches zero, the gradient decreases in the case of ℓ_2—approaching zero as our model parameters approach zero—but remains constant in the case of ℓ_1. This means that no matter how close or distant our model parameter vector is to zero, regularization via the ℓ_1-norm applies the same magnitude of update towards zero.*

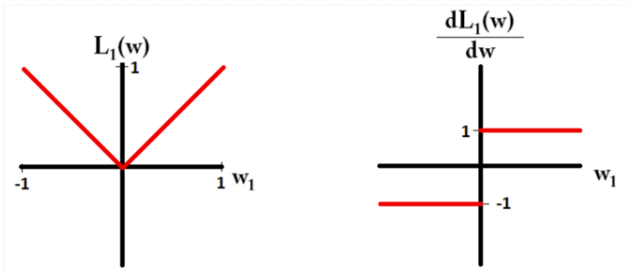

Figure 4: The ℓ_1-norm and its gradient

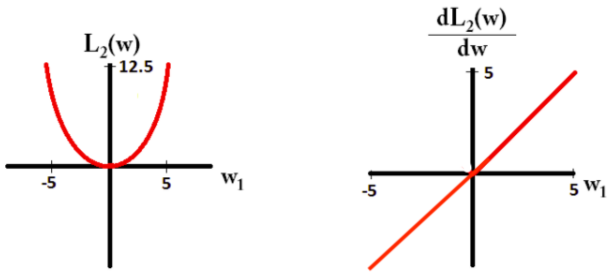

Figure 5: The ℓ_2-norm and its gradient