Multinomial classification

A Multinomial Classifier (when categories/classes $||C|| > 2$) can be created from multiple Binary Classifiers

- Create a separate Binary Classifier for each $c \in C$
- The classifier for category c attempts to classify
    - Each example with target category of c as Positive
    - All other examples as Negative
- Combine the $||C||$ classifiers to produce a vector $\hat{p}$ of length $||C||$
    - normalize across $c \in C$ to sum to 1
    - $\hat{p}_c$ denotes the normalized value for category c
        - **Notation abuse**: subscripts should be integers, not categories

- **This is not something you need to do for yourself**
    - Built into `sklearn`
    - "All classifiers in scikit-learn do multiclass classification out-of-the-box."
    - sklearn.multiclass.OneVsRestClassifier(estimator) if you want to create your own binary `estimator`

Cross Entropy: Loss function for multinomial classification

Both the target $\mathbf{y}$ and the prediction $\hat{p}$ are represented as vectors of length $||C||$

- We write $\mathbf{y}_c, \hat{p}_c$ to denote the element of the vector corresponding to category c
- Each vector can be interpreted as a probability distribution, e.g.
$$\forall c \in C : \mathbf{y}_c \geq 0$$
$$\sum_{c \in C} \mathbf{y}_c = 1$$
    - $\mathbf{y}$ was created with One Hot Encoding (OHE), so properties satisfied by consruction
    - $\hat{p}_c$ satisfies the properties by virtue of the normalization of the predictions of the $||C||$ binary classifiers

With $\mathbf{y}, \hat{p}$ encoded as a vectors, per example Binary Cross Entropy can be generalized to $||C|| \geq 2$ categories:

$$\mathcal{L}_{\Theta}^{(\mathbf{i})} = -\sum_{c=1}^{||C||} \left(\mathbf{y}_c^{(\mathbf{i})} * \log(\hat{\mathbf{p}}_c^{(\mathbf{i})}) \right)$$

This is the multinomial analog of Binary Cross Entropy and is called **Cross Entropy**.

Cross Entropy can be interpretted as a measure of the "distance" between distributions $\mathbf{y}$ and $\hat{p}$

- Minimized when they are identical
- We will use Cross Entropy in the future both as a Loss function and a way of comparing probability distributions

Accuracy is *not* a loss function

Recall the mapping of probability to prediction

$$\hat{y}^{(\mathbf{i})} = \begin{cases} \text{Negative} & \text{if } \hat{p}^{(\mathbf{i})} < 0.5 \\ \text{Positive} & \text{if } \hat{p}^{(\mathbf{i})} \geq 0.5 \end{cases}$$

The prediction for example i changes only when probability $\hat{p}^{(\mathbf{i})}$ crosses the threshold.

Suppose the class for example i is Positive: $\mathbf{y}^{(\mathbf{i})} = \text{Positive}.$

- Is our model "better" when

$$
\begin{array}{llll}
\hat{p}^{(\mathbf{i})} & \approx 1 & \text{than when} & \hat{p}^{(\mathbf{i})} = 0.5 \\
\hat{p}^{(\mathbf{i})} & = (.5 - \epsilon) & \text{than when} & \hat{p}^{(\mathbf{i})} \approx 0
\end{array}
$$

- The per-example Accuracy is the same in both comparisons
- But a model with probability $\hat{p}^{(\mathbf{i})}$ as close to $\mathbf{y}^{(\mathbf{i})} = 1$ would seem to better

There is no *degree* or magnitude of inaccuracy

- Two models may have the same Accuracy even though the probabilities of one may be closer to perfect than the other
- In our search for the best Θ, Accuracy won't be a guide

Recap of week 3

- Good news
    - You now know two main tasks in Supervised Learning
        - Regression, Classification
    - You now know how to use virtually every model in `sklearn`
        - Consistent API
            - `fit`, `transform`, `predict`
    - You survived the "sprint" to get you up and running with ML
    - You know the *mechanical process* to implement transformations: Pipelines

- Even better news
    - There's a lot more !
        - Machine Learning is about *problem solving* **not** using an API

This week we start to focus on how to become an effective problem solver

- Error Analysis
- Deeper Understanding of Loss functions
- Transformations in more depth

All of this is in service of how you can improve models

In [2]:

```
print("Done")
```

Done