ChatGPT

Classification Concepts

Binary Classification

Binary classification is the task of assigning each input to one of two classes (often labeled 0 and 1, or positive and negative). In a binary classification problem, the model learns a decision rule that maps inputs x to a discrete label $y \in \{0, 1\}$ [1]. A common approach is to use a linear function $h(x) = w^\top x$ (a *discriminant function*) whose **sign** indicates the predicted class (e.g. if $w^\top x > 0$ predict class 1, else class 0) [2]. The **magnitude** of $w^\top x$ can be interpreted as the confidence in the prediction. In logistic regression (a typical binary classifier), this linear output is fed through a *sigmoid* to obtain probabilities, making the model easier to train [3] [4]. Binary classification models are evaluated using metrics like accuracy, precision, and recall that are based on the counts of true positives, false positives, etc.

Performance Metrics

Performance metrics quantify how well a classifier is doing, typically using the counts of true/false positives/negatives. There are four fundamental outcomes for each example: **True Positive (TP)** (correctly predicted positive), **False Positive (FP)** (negative predicted as positive), **True Negative (TN)** (correctly predicted negative), and **False Negative (FN)** (positive predicted as negative) [5]. These counts are often organized into a *confusion matrix*, a table whose rows are the true class and columns are the predicted class [6]. From the confusion matrix one computes metrics such as:

- **Accuracy** = (TP + TN) / (TP + FP + TN + FN). It measures the overall fraction of correctly classified examples [7].
- **Precision** = TP / (TP + FP). It is the fraction of positively predicted examples that are actually positive [7].
- **Recall (Sensitivity)** = TP / (TP + FN). It is the fraction of actual positives that were correctly identified [7].
- **F1 Score** = $2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$, the harmonic mean of precision and recall [7].

These metrics capture different aspects of performance; for instance, F1 balances precision and recall, which is useful when classes are imbalanced [7]. (Accuracy alone can be misleading on imbalanced data, so one often uses precision/recall/F1 instead [8] [7].)

Multiclass Classification

In multiclass classification, there are more than two classes (say $C > 2$), and each example's label y can take values $\{0, 1, \ldots, C-1\}$. The model typically outputs a probability for each class. A common generalization of logistic regression is to have a separate weight vector w_c for each class c, and compute unnormalized scores $s_c = w_c^\top x$ for each class. To convert these scores into valid class probabilities p_c that sum to 1, one uses the **softmax** function:

$$\sigma(s)_c \;=\; \frac{e^{s_c}}{\sum_{j=1}^{C} e^{s_j}}.$$

This ensures $p_1 + \cdots + p_C = 1$ [9]. The predicted class is typically the one with highest probability. (A "hardmax" or argmax decision rule would assign probability 1 to the highest-scoring class and 0 to others.) Learning is done by minimizing the *cross-entropy loss*, which for a training set $\{(x_i, y_i)\}$ is:

$$L_{CE}(W) = -\sum_{i=1}^{m} \log \sigma(w_{y_i}^\top x_i)\,,$$

where w_{y_i} is the weight vector of the true class for example i [10]. This loss encourages the model to assign high probability to the correct class. Multiclass versions of performance metrics can also be defined via one-vs-all confusion matrices for each class.

SoftMax and Hardmax

The **softmax** function (described above) maps a vector of real scores to a probability distribution over classes [9]. Each class gets a positive probability proportional to e^{score}, so even non-maximal classes get some probability. In contrast, a **hardmax** (sometimes just argmax) is a non-differentiable operation that assigns full weight to the highest-scoring class: it outputs a one-hot vector that has 1 for the class with the largest score and 0 for all others. The softmax is useful in training (it is smooth and differentiable) while hardmax represents the final discrete decision. In practice, one trains with softmax probabilities and then at prediction time usually just picks the class with maximum probability (effectively performing a hardmax on the learned probabilities).

Sigmoid Function

The **sigmoid** function is a smooth, S-shaped curve defined by $\sigma(u) = 1/(1 + e^{-u})$. In logistic regression for binary classification, the raw linear score $u = w^\top x$ is passed through the sigmoid to produce a probability $p = \sigma(u)$ of class 1 [4]. The sigmoid maps real-valued scores to the interval $(0, 1)$, with $\sigma(0) = 0.5$. Its derivative $\sigma'(u) = \sigma(u)(1 - \sigma(u))$ is always positive, which makes it suitable for gradient-based training. The inverse of sigmoid is the logit or log-odds: $\log(p/(1-p)) = u$, which shows why $u = w^\top x$ can be interpreted as the log-odds of the positive class [4]. The sigmoid thus converts a linear discriminant into a probability estimate, allowing optimization of differentiable loss functions.

Cross Entropy Loss

Cross-entropy loss measures how well a probabilistic classifier's predictions match the true labels. For binary logistic regression, the loss for one example with predicted probability p and true label $y \in \{0, 1\}$ is

$$-\big[y \log(p) + (1 - y) \log(1 - p)\big],$$

and for a dataset one sums over all examples [11]. For multiclass classification with softmax outputs $\sigma(\cdot)$, the multiclass cross-entropy loss is

$$-\sum_{i=1}^{m} \log \sigma(w_{y_i}^\top x_i),$$

where y_i is the true class of example i [10]. Intuitively, this loss is the negative log-likelihood of the correct classes under the model's predicted probability distribution. Minimizing cross-entropy encourages the model to assign high probability to the true class (since $-\log(p)$ is large when p is

small). The term “cross-entropy” comes from information theory, measuring the distance between the true label distribution and the model’s predicted distribution.

K-Fold Cross Validation

K-Fold cross validation is a technique to more reliably estimate a model’s performance by averaging over multiple train-test splits [12]. One splits the data into K roughly equal *folds*. Then, for each $k = 1, \ldots, K$, the model is trained on $K-1$ folds and tested on the remaining fold. This process yields K performance measurements (e.g. accuracy) on different held-out sets. The final reported metric is the average of these K values [12]. This helps reduce the dependency of the performance estimate on any particular random split, which is important because a single train/test split can give a misleading result if, say, one class is underrepresented in that split.

Receiver Operating Characteristic (ROC)

A **ROC curve** visualizes the trade-off between true positive rate and false positive rate as one varies the classification threshold. Formally, it plots **True Positive Rate (TPR)** against **False Positive Rate (FPR)** for all possible threshold values on the classifier’s score. The true positive rate is $TPR = TP/(TP + FN)$ and the false positive rate is $FPR = FP/(FP + TN)$. As we sweep the threshold from low to high, we move along the ROC curve. An ROC curve that bows toward the top-left indicates good performance. The **Area Under the ROC Curve (AUC)** is a single-number summary of the ROC curve: it is the probability that the classifier will rank a random positive example higher than a random negative one. A perfect classifier has $AUC = 1.0$, while random guessing yields $AUC = 0.5$ [13]. Thus, higher AUC indicates a stronger classifier.

AUC (Area Under the Curve)

The **AUC** is defined as the area under the ROC curve, which ranges between 0 and 1 [13]. It captures the model’s ability to discriminate positive from negative examples across all thresholds. As noted, an AUC of 1 means perfect discrimination (all positives rank above all negatives), while an AUC of 0.5 corresponds to no discriminatory power beyond random chance [13]. Practically, AUC is used as a robust metric especially when classes are imbalanced, because it does not depend on choosing one particular threshold.

Confusion Matrix

A **confusion matrix** tabulates the counts of actual versus predicted classes for a classifier [6]. For binary classification, it is a 2×2 table: the rows are the true class (positive/negative) and the columns are the predicted class (positive/negative). Each cell contains the number of examples falling into that combination. For example, the top-left cell (True Positive) is the count of positives correctly predicted as positive, and so on. The confusion matrix (when populated with counts) allows quick computation of accuracy, precision, recall, etc., for any class. In multiclass problems, one can similarly form a confusion matrix of size $C \times C$, where the diagonal entries are correct predictions. In summary, the confusion matrix summarizes all possible outcomes of classification, making it easy to compute the performance metrics [6].

Accuracy, Precision, F1 Score, Recall

These standard performance metrics are all derived from the confusion matrix counts. **Accuracy** is the overall fraction correct: $(TP + TN)/(TP + TN + FP + FN)$. **Precision** is $TP/(TP + FP)$, the fraction of positive predictions that were actually correct. **Recall** (also called sensitivity or true positive rate) is $TP/(TP + FN)$, the fraction of actual positives that were identified. The **F1 score** is the harmonic mean of precision and recall: $F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$ [7] . The F1 score is useful because it balances precision and recall: for example, if either precision or recall is low, the F1 score will also be low. These formulas and definitions come directly from the entries of the confusion matrix [7] . (In imbalanced settings where one class is rare, precision and recall are often more informative than accuracy.)

K-Nearest Neighbors (KNN)

The *k-nearest neighbors* classifier is a simple non-parametric method: to classify a new point x , one finds the k closest points in the training set (according to some distance metric) and assigns x to the majority class among those neighbors [14] . Because it simply "remembers" the training data and uses voting of neighbors, kNN does not have a fixed set of parameters to learn – this makes it a **non-parametric classifier** [15] . The choice of k is a hyperparameter: small k can lead to a very wiggly boundary (overfitting), while large k smooths the boundary. One can also weight the neighbors by inverse distance so that closer neighbors have more influence on the vote. The decision boundary of kNN is generally complex: for 1-NN, each training point has a *Voronoi cell* (region of space closer to that point than any other) that is classified as that point's class [16] . A drawback of kNN is that prediction time can be slow (linear in the training set size) since it must compute distances to all points to find the neighbors.

Logistic Regression

Logistic regression is a popular method for binary classification. It models the log-odds of the positive class as a linear function of the inputs: $\log(p/(1-p)) = w^\top x$. Equivalently, it applies the sigmoid to $w^\top x$ to produce the probability p of class 1 [4] . In practice, logistic regression is trained by maximizing likelihood (or minimizing cross-entropy loss) so that it assigns high probability to the correct labels. Despite its name, logistic regression is actually a classification algorithm. It is popular because it is simple, yields interpretable linear decision boundaries, and its probabilistic output facilitates calibration and thresholding. In the multiclass setting, logistic regression naturally generalizes via the softmax approach described above.

Discriminant Function

A *discriminant function* in classification is a real-valued function $h(x)$ whose sign (or largest component) is used to decide the class label [2] . In linear classification, we use $h(x) = w^\top x$. For a binary problem, $\text{sign}(w^\top x)$ determines the predicted class: positive sign for class 1, negative for class 0 [2] . In multiclass problems, one often uses a vector of discriminant functions (one per class) and chooses the class with the highest score. The discriminant function is essentially the model's raw output before converting to probabilities. In logistic regression, $w^\top x$ is the log-odds score, but one could use it directly for decisions if one ignores probability calibration. Using a discriminant function allows a classifier to have linear decision boundaries (hyperplanes) in the input space.

Non-Parametric Classifier

A **non-parametric classifier** is one that does not assume a fixed finite set of parameters for modeling, but rather its complexity can grow with the amount of data. The kNN classifier is a classic example: it effectively "stores" all training examples and makes predictions based on the nearest ones [15]. Since kNN's decision rule depends on the entire dataset (not just a fixed-weight vector), it is non-parametric. Another way to say this is that a non-parametric model's number of parameters can grow with the training set size. Non-parametric models are often very flexible (high capacity) and can fit complex patterns, but they can be slower at prediction and may require more data to generalize well. In contrast, models like logistic regression are parametric (fixed-size parameter vector w of dimension equal to the number of features) [15].

Voronoi Cell

In geometry and classification, the **Voronoi cell** of a training point x is the set of all query points that are closer to x than to any other training point [16]. The collection of all Voronoi cells (one per training point) partitions the input space. In a 1-nearest neighbor (1-NN) classifier, each cell is assigned the class of its generating point, so the decision boundary lies along the edges of the Voronoi diagram. For example, in 1-NN with two points of different classes, the decision boundary is the perpendicular bisector of the segment connecting them (the set of points equidistant to both) [16]. For k-NN with $k > 1$, the boundary is similarly determined by loci of points that have ties in nearest neighbors. Voronoi cells are a useful geometric concept to visualize how instance-based classifiers like kNN partition the space.

Sources: Explanations above are based on the provided notebook *03_Classification.ipynb*, specifically its discussion of logistic regression, performance metrics, ROC/AUC, k-fold validation, kNN, and related concepts [1] [17] [13] [12]. These references contain formal definitions and formulas for the concepts discussed.

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] 03_Classification.ipynb
file://file-DAQni7JDvoTAZ9dVBJiFwZ