Google Colab is a free online Jupyter notebook environment that allows you to run and experiment with machine learning models. It's a great platform for quickly prototyping and testing ideas, as it provides access to a variety of powerful computing resources, including GPUs and TPUs, which can significantly speed up the training process.

To start using Google Colab, you'll first need to sign in to your Google account. If you don't already have a Google account, you can create one for free at https://accounts.google.com/signup.

Once you're signed in, you can access Google Colab by going to https://colab.research.google.com/. This will open the Google Colab homepage, which provides a list of your recent notebooks and access to the documentation and tutorials.

To create a new notebook, click the "New Notebook" button in the top-right corner of the homepage. This will open a new tab with an empty notebook, where you can start writing and executing code.

Alternatively, you can also use Google Drive to create a Colab notebook. To do this, go to https://drive.google.com/, and click the "New" button in the top-left corner. From the dropdown menu, select "More" and then "Google Colab". This will create a new Colab notebook and open it in a new tab.

Few classification algorithms to try on sets of handwritten digits:
1. support vector machines (SVMs)
2. k-nearest neighbors (KNNs)
3. convolutional neural networks (CNNs)
4. Random forest

Support Vector Machines (SVMs): SVMs are a type of supervised learning algorithm that can be used for classification and regression tasks. They work by finding the hyperplane in a high-dimensional space that maximally separates the different classes. SVMs are effective when the data is linearly separable, but can be extended to nonlinear problems using techniques such as the kernel trick.

K-Nearest Neighbors (KNNs): KNNs is a simple and effective classification algorithm that works by finding the K nearest data points to a given test point and assigning the label of the majority of these points to the test point. KNNs are often used for image classification because they are easy to implement and require little training data.

Convolutional Neural Networks (CNNs): CNNs are a type of neural network that is particularly well-suited for image classification tasks. They are designed to automatically learn features from the input data, and can achieve state-of-the-art results on many image

classification benchmarks. CNNs are often trained using large datasets and require significant computational resources, but are widely used in industry due to their effectiveness.

Random Forests: Random forests are an ensemble learning method that can be used for classification and regression tasks. They work by training a large number of decision trees on random subsets of the data and combining their predictions through a majority vote. Random forests are effective for many types of data, including images, and are relatively easy to use.

Date: 12/01/2023

Gaussian Naive Bayes algorithm for classification

Gaussian Naive Bayes (GNB) is a type of probabilistic classifier based on the Bayes theorem. It is called "Naive" because it makes the assumption that all the features in a dataset are independent of each other, which is usually not the case in real-world datasets. Despite this strong assumption, GNB has been found to work well in practice for many types of datasets, particularly for datasets with continuous features.

The basic idea behind the GNB algorithm is to model the probability density function (PDF) of each feature for each class. Since GNB assumes that the features are normally distributed, it models the PDF of each feature as a Gaussian distribution. The classifier then uses these estimated PDFs to predict the class of new instances.

The GNB algorithm can be summarized in the following steps:

1. Estimate the mean and standard deviation of each feature for each class using the training data.
2. For a new instance, calculate the probability of the instance belonging to each class using the Gaussian PDFs of the features and the Bayes theorem.
3. Predict the class with the highest probability.

The GNB algorithm is known for its simplicity and efficient implementation. It is also easy to interpret and understand the results, which makes it a good choice for applications where interpretability is important. It's also less computationally expensive than other algorithms.

It's worth noting that Gaussian Naive Bayes is a generative model, which means that it models the probability of the data generating the model. On the other hand, other algorithms like logistic regression or support vector machines are discriminative models, which means that they model the decision boundary between classes.

Here is the code using GNB IRIS flower dataset:

https://colab.research.google.com/drive/1OTslBe-xLxr_hhtmmdXfRN2gEFUeTp3F?usp=sharing

Evaluation metrics:

True Positives e.g for class A
→ how many objects in A are correctly classified as belonging to class A

False Positives
→ how many objects from other class are (incorrectly) assigned classified into class A

True Negatives
→ how many objects not in A are classified as not in A (in binary classification with only two classes A & B then replace 'not in A' to 'class B')

False Negative
→ how many objects from class A has been misclassified as some other class

(True Negative and False Negative are basically same as True Positive and True Negative for class B if the classification is binary)

Explaining more with examples:

For example, consider a binary classification problem where the positive class represents the presence of certain disease and the negative class represents the absence of that disease.

A false positive (FP) would be an instance where the classifier predicts the presence of the disease, but the patient actually does not have the disease. In other words, it's a case where the classifier incorrectly classifies a negative instance as positive. A false positive can be thought of as a "false alarm", and it can have serious consequences in some cases, such as unnecessary treatments or further testing.

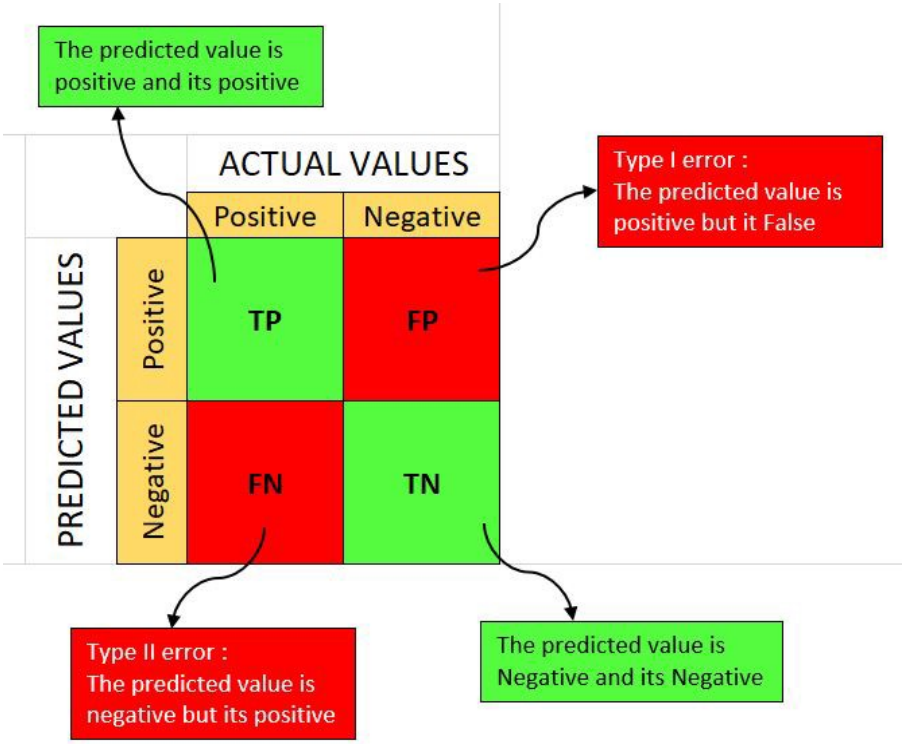
On the other hand, a false negative (FN) would be an instance where the classifier predicts the absence of the disease, but the patient actually has the disease. In other words, it's a case where the classifier incorrectly classifies a positive instance as negative. A false negative can be thought of as a "missed detection", and it can also have serious consequences, such as a disease going undiagnosed.

In the context of this example, true positives (TP) would be the number of instances where the classifier correctly predicts the presence of the disease, true negatives (TN) would be the number of instances where the classifier correctly predicts the absence of the disease, false

positives (FP) would be the number of instances where the classifier incorrectly predicts the presence of the disease, and false negatives (FN) would be the number of instances where the classifier incorrectly predicts the absence of the disease.

It's worth noting that the cost of false positives and false negatives can vary depending on the context, and the appropriate metric of evaluation should be chosen accordingly.

For example, if missed diagnostic has to be reduced then one can design two tests: in the first one should minimize false negatives, and then perform the second test on all positive cases. In the second test then one should minimize the false positives. This is what is usually done for life-threatening diseases.



A confusion matrix:

A confusion matrix, also known as an error matrix, is a table that is used to define the performance of a classification algorithm. It is a way to visualize the performance of a classifier by comparing the predicted class labels with the true class labels.

A confusion matrix is typically represented as a table with the following four entries: True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN)

Here (and in image above) is an example of a confusion matrix for a binary classification problem:

	Actual Positives	Actual negatives
Predicted Positives	TP	FP
Predicted Negatives	FN	TN

With the help of the confusion matrix, you can calculate many important evaluation metrics such as accuracy, precision, recall, F1-score, which are defined as:

- Accuracy: (TP + TN) / (TP + TN + FP + FN)
- Precision: TP / (TP + FP)
- Recall: TP / (TP + FN)
- F1-score: 2 * (Precision * Recall) / (Precision + Recall)

It's important to note that these evaluation metrics are not always appropriate for all classification problems, as they have different properties, and the most suitable metric will depend on the characteristics of the dataset and the goals of the analysis.

In addition, for multi-class classification problems, you can also create a multi-class confusion matrix that will contain more than two classes.

More on the accuracy, precision, recall, and F1 scores:

Accuracy:
Accuracy is a metric used to evaluate the performance of a classification model. It is defined as the ratio of correctly classified instances to the total number of instances. Mathematically, it can be defined as the following:

Accuracy = (Number of Correct Predictions) / (Total Number of Predictions)

For example, if a classifier is trained on a dataset containing 100 instances and it correctly classifies 80 of them, then its accuracy is 80%.

Accuracy is a widely used metric and it is simple to understand and interpret, but it can be misleading in some cases. For example, if the dataset has imbalanced classes (i.e. one class has much more instances than other classes), accuracy can be a poor metric to evaluate the performance of the model since it might be able to achieve a high accuracy by simply predicting the majority class for all instances.

To overcome this problem, other metrics like precision, recall and F1-score can be used. These metrics take into account the number of false positives and false negatives, and are more informative when the classes are imbalanced.

Precision:
Precision is a metric used to evaluate the performance of a classification model, it is a measure of the ability of the model to correctly classify instances of the positive class. It is defined as the ratio of true positive predictions to the total number of positive predictions. Mathematically, it can be defined as the following:

Precision = (True Positives) / (True Positives + False Positives)

For example if a classifier is trained on a dataset containing 100 instances and it correctly classifies 80 of the positive instances (True Positives) and incorrectly classifies 10 negative instances as positive (False Positives), then its precision is 80 / (80 + 10) = 0.8.

Precision is a good metric to use when the goal of the model is to avoid false positives, for example in medical diagnosis, where a false positive could mean a patient undergoes unnecessary treatment. In other cases, where missing true positives is more costly than a high number of false positives, recall is a more appropriate metric.

It's worth noting that precision and recall are trade-off with each other and they are both important to evaluate a classifier. Therefore, it is common to use F1-score which is the harmonic mean of precision and recall and it is a good balance between these two metrics.

– This (text below) will be covered in the introduction

Here are some topics that you may want to familiarize yourself with before writing a simple classification machine-learning code:

Data preprocessing: You'll need to understand how to clean and prepare your data for the machine learning model. This may include tasks such as handling missing values, encoding categorical variables, and scaling numeric features.

Classification algorithms: There are many different classification algorithms to choose from, each with its own strengths and weaknesses. It's a good idea to familiarize yourself with the most commonly used algorithms, such as **logistic regression, k-nearest neighbors, decision trees, and support vector machines**, so that you can choose the one that is most suitable for your problem.

Evaluation metrics: You'll need to understand how to evaluate the performance of your classification model. Common evaluation metrics for classification include accuracy, precision, recall, and f1 score.

Overfitting and underfitting: It's important to understand the concepts of overfitting and underfitting, as these can affect the generalization performance of your model. Overfitting occurs when a model is too complex and starts to memorize the training data, while underfitting occurs when the model is too simple and cannot capture the underlying patterns in the data.

Hyperparameter tuning: Many classification algorithms have hyperparameters that can be tuned to improve the model's performance. You'll need to understand how to select appropriate values for these hyperparameters and how to use techniques such as grid search or random search to find the best combination of hyperparameters for your problem.