Classification on Colab using MNIST dataset

Submitted by Manish Bafna
Student Id: 19655
Instructor: Dr. Henry Chang
GIT: Classification using MNIST dataset

Table of Content

Introduction

The most common supervised learning tasks are

- Regression (predicting values)
  - Linear Regression
  - Non-linear Regression
    - Decision Trees
      - Random Forests
- Classification (predicting classes)

Introduction

MNIST dataset is a set of 70,000 small images of handwritten digits. Scikit-Learn is a Machine Learning Library which provides many helper functions to download popular datasets, MNIST is one of them.

Instead of recognize 9 digits

- Let's simplify the problem for now

  and only try to identify one digit
    - For example, the number 5.

Introduction

Classification Model Evaluation Metric: Confusion Matrix

Suppose you're working as a security guard at a concert and your job is to

- identify any individuals who were troublemakers at previous events (represented as "positive" class in this problem).

Recall	F1 Score

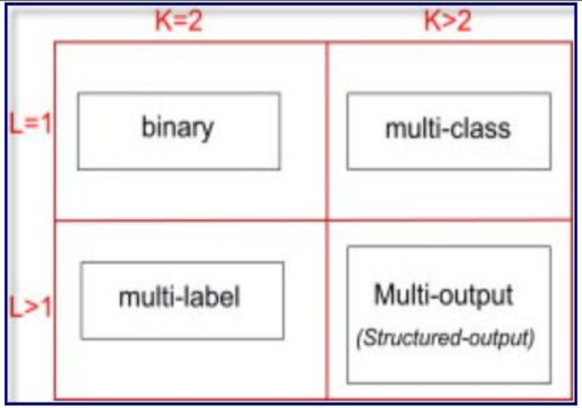
Recall measures the proportion of true positive instances (troublemakers) that were correctly identified by the classifier.
• The fraction of actual troublemakers that were correctly identified by you as troublemakers and allowed into the concert.
	F1 Score is a balance between
	• Precision and
	  ◦ Precision is the fraction of instances classified as positive that are actually positive,
	• Recall
	  ◦ Recall is the fraction of positive instances that were correctly classified as positive.

A high recall score means that you are able to catch most of the troublemakers, but you may also let in some non-troublemakers (false positives).	In this analogy, F1 Score would represent the balance between catching all the troublemakers (high recall) and only letting in the non-troublemakers (high precision). ◦ A high F1 Score means that you have found a good balance between these two metrics.

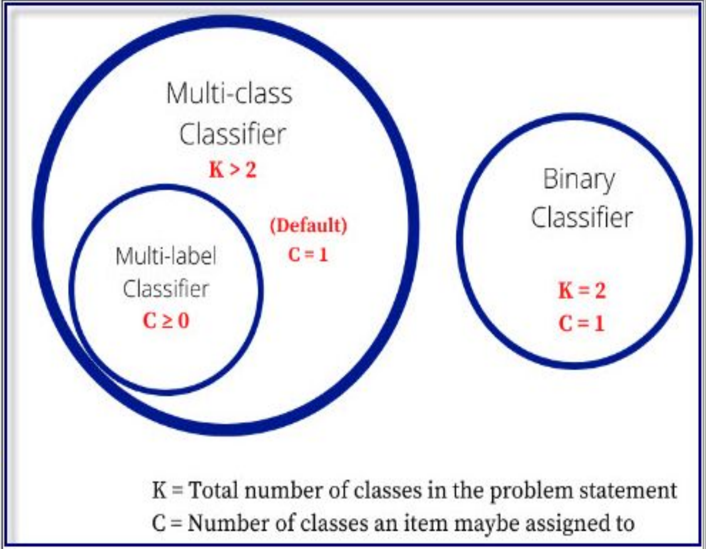
Introduction

Types of Classification

Summary



Scalable multi-output label prediction: From classifier chains to classifier trellises



Multiclass v/s Multilabel classification

Implementation

- This “5-detector” will be an example of a binary classifier, capable of distinguishing between just two classes, 5 and not-5.
- Let’s create the target vectors for this classification task:
  y_train_5 = (y_train == 5) # True for all 5s, False for all other digits
- y_test_5 = (y_test == 5)

- Now let’s pick a classifier and train it.
  - A good place to start is with a Stochastic Gradient Descent (SGD) classifier, using Scikit-Learn’s SGDClassifier class.
  - This classifier has the advantage of being capable of handling very large datasets efficiently.
  - This is in part because SGD deals with training instances independently, one at a time (which also makes SGD well suited for online learning), as we will see later.

Implementation

Let’s create an SGDClassifier and train it on the whole training set:

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)

sgd_clf.fit(X_train, y_train_5)
```

- The SGDClassifier relies on randomness during training (hence the name “stochastic”).
  - If you want reproducible results, you should set the random_state parameter.
- Now we can use it to detect images of the number 5:

```
>>> sgd_clf.predict([some_digit])

array([ True])
```

- The classifier guesses that this image represents a 5 (True).
  - Looks like it guessed right in this particular case!

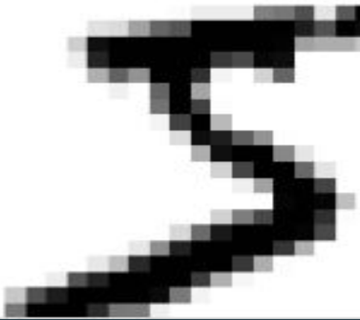
Implementation

```
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt

some_digit = X[0]
some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap=mpl.cm.binary)
plt.axis("off")

save_fig("some_digit_plot")
plt.show()
```

```
Saving figure some_digit_plot
```



Implementation

```
plt.figure(figsize=(9,9))
example_images = X[:100]
plot_digits(example_images, images_per_row=10)
save_fig("more_digits_plot")
plt.show()
```

Implementation

Use Google Colab

Understanding MNIST database

Run the 03_classification.ipynb on google colab

Test

Measuring Accuracy Using Cross-Validation

```
[18] from sklearn.model_selection import StratifiedKFold
     from sklearn.base import clone

     skfolds = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

     for train_index, test_index in skfolds.split(X_train, y_train_5):
         clone_clf = clone(sgd_clf)
         X_train_folds = X_train[train_index]
         y_train_folds = y_train_5[train_index]
         X_test_fold = X_train[test_index]
         y_test_fold = y_train_5[test_index]

         clone_clf.fit(X_train_folds, y_train_folds)
         y_pred = clone_clf.predict(X_test_fold)
         n_correct = sum(y_pred == y_test_fold)
         print(n_correct / len(y_pred))

     0.9669
     0.91625
     0.96785
```

Test

Confusion Matrix

```
[23] y_train_perfect_predictions = y_train_5  # pretend we reached perfection
     confusion_matrix(y_train_5, y_train_perfect_predictions)

array([[54579,     0],
       [    0,  5421]])
```

Test

Confusion Matrix

Precision and Recall:

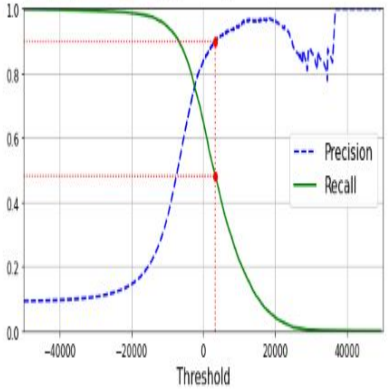
```
[36] def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
         plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
         plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
         plt.legend(loc="center right", fontsize=16) # Not shown in the book
         plt.xlabel("Threshold", fontsize=16)        # Not shown
         plt.grid(True)                              # Not shown
         plt.axis([-50000, 50000, 0, 1])             # Not shown



     recall_90_precision = recalls[np.argmax(precisions >= 0.90)]
     threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]


     plt.figure(figsize=(8, 4))                                                         # Not shown
     plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
     plt.plot([threshold_90_precision, threshold_90_precision], [0., 0.9], "r:")        # Not shown
     plt.plot([-50000, threshold_90_precision], [0.9, 0.9], "r:")                       # Not shown
     plt.plot([-50000, threshold_90_precision], [recall_90_precision, recall_90_precision], "r:")# Not shown
     plt.plot([threshold_90_precision], [0.9], "ro")                                    # Not shown
     plt.plot([threshold_90_precision], [recall_90_precision], "ro")                    # Not shown
     save_fig("precision_recall_vs_threshold_plot")                                     # Not shown
     plt.show()
```

Saving figure precision_recall_vs_threshold_plot



Test

```
[86] ambiguous_digit = X_test[2589]
     knn_clf.predict_proba([ambiguous_digit])

     array([[0.24579675, 0.        , 0.        , 0.        , 0.        ,
             0.        , 0.        , 0.        , 0.        , 0.75420325]])

[87] plot_digit(ambiguous_digit)
```

Enhancement

Hyperparameter Tuning: The performance of KNN can be greatly improved by fine-tuning the hyperparameters such as the number of neighbors, the distance metric, etc. We can use techniques such as grid search or random search to find the optimal hyperparameters.

Anomaly Detection: We can use KNN for anomaly detection on MINST, where the goal is to identify images that are significantly different from the rest of the data.

Feature Engineering: Feature engineering is an important step in enhancing the performance of KNN. We can try different features such as edge detection, shape detection, histograms, etc.

Conclusion

In conclusion, KNN is a simple yet powerful classification algorithm that can be applied to the MINST dataset.

By following best practices such as data preprocessing, feature engineering, dimensionality reduction, hyperparameter tuning, and ensemble methods, we can significantly improve the performance of KNN on MINST.

However, it's important to remember that there is no one-size-fits-all solution, and the best approach may vary based on the specific use case and requirements. It's always a good idea to experiment with different approaches and evaluate their performance to find the best solution for the problem.

References

3 Classification.in (sfbu.edu)

03_classification.ipynb - Colaboratory (google.com)