CLASSFICATION ANALYSIS ON MNIST DATASET

Introduction

In this report, we present a classification analysis on the MNIST dataset to determine whether a given image represents the number 3 or not. We employed three different classifiers: Support Vector Classifier (SVC), Random Forest, and SGDClassifier. The objective of this analysis is to evaluate the performance and compare the predictive capabilities of these classifiers on the given task.

Dataset Overview

The MNIST dataset is a popular benchmark dataset in machine learning, consisting of 70,000 grayscale images of handwritten digits. Each image is a 28x28 pixel representation of a digit ranging from 0 to 9. For our analysis, we focused on classifying images as either representing the number 3 or not.

Data Preprocessing and Feature Engineering

Before building the classifiers, we performed some data preprocessing and feature engineering steps. This included splitting the dataset into training and test sets, as well as encoding the target variable to identify whether the digit is equal to 3 or not.

Classification Models

a. SGDClassifier:

First, we utilized the SGDClassifier to classify the digits as 3 or not. We trained the model using the Stochastic Gradient Descent (SGD) algorithm with default hyperparameters. We made predictions on the test set and evaluated the model's performance using various metrics.

b. Random Forest Classifier

Next, we built a Random Forest classifier to perform the classification task. We used the Random Forest algorithm with default hyperparameters and trained the model on the training set. We then made predictions on the test set and evaluated the model's performance using various metrics.

c. Support Vector Classifier (SVC)

We finally trained an SVC model on the MNIST dataset to classify whether the digit is equal to 3 or not. We used the default hyperparameters and employed the fit method to train the model on the training set. We then made predictions on the test set and evaluated the model's performance using various metrics.

Model Evaluation and Comparison

We evaluated the performance of each classifier using the following metrics:

- Accuracy Score
- Precision
- Recall
- F1 Score
- Confusion Matrix
- Cross-Validation Score

Based on the evaluation results, we compared the classifiers in terms of their accuracy, precision, recall, and F1 score. We also analyzed the confusion matrices to gain insights into the classifier's performance on different classes.

Key Findings and Insights

Through the classification analysis, we obtained the following key findings and insights:

i. The Random Forest classifier demonstrated the highest F1 score of 0.908, indicating its superior performance in correctly classifying whether a digit is equal to 3 or not. This classifier achieved a good balance between precision and recall, making it a reliable choice for this task.

ii. The SGDClassifier showed a moderate F1 score of 0.82, indicating reasonably good performance but slightly lower than the Random Forest classifier. While it may not perform as well as the Random Forest classifier, it still provides acceptable accuracy for classifying the digits.

iii. The SVC model achieved the lowest F1 score of 0.96 among the three classifiers. Although it had a relatively high accuracy, it showed slightly lower precision and recall compared to the Random Forest classifier. Nevertheless, it still performed well in distinguishing between the digit 3 and others.

iv. The results suggest that the Random Forest classifier is the most suitable choice for this specific classification task, considering its higher F1 score and overall performance. It strikes a good balance between precision and recall, providing accurate predictions for classifying whether a digit is equal to 3 or not.

v. The findings indicate that different classifiers can yield varying results in terms of accuracy and predictive capabilities. It is crucial to select the appropriate classifier based on the specific requirements and priorities of the task at hand.

Suggestions for Next Steps

- To further improve the classification analysis on the MNIST dataset, we recommend considering the following steps:
- Explore additional feature engineerings techniques such as scaling, dimensionality reduction, or feature extraction to enhance model performance.
- Experiment with different hyperparameter configurations for each classifier to optimize their performance.
- Consider ensemble methods such as voting or stacking to combine the predictions of multiple classifiers for improved accuracy and robustness.

Conclusion

In conclusion, the classification analysis on the MNIST dataset revealed that the SVC model achieved the best performance for determining whether a digit is equal to 3 or not. With an F1 score of 0.96, the SVC model demonstrated high accuracy and precision in classifying the target class. The findings emphasize the importance of selecting the appropriate classifier for specific tasks, and in this case, the SVC model stands out as the recommended choice for accurate digit classification.

Python Implementation

clf-project

July 3, 2023

0.1 Classification Project

0.1.1 Load Data

[29]:
```
from sklearn.datasets import fetch_openml
```

[30]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
```

[31]:
```
mnist = fetch_openml('mnist_784', as_frame = False);
```

```
c:\Users\nhatk\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\datasets\_openml.py:1002: FutureWarning: The default value of
`parser` will change from `'liac-arff'` to `'auto'` in 1.4. You can set
`parser='auto'` to silence this warning. Therefore, an `ImportError` will be
raised from 1.4 if the dataset is dense and pandas is not installed. Note that
the pandas parser may return different data types. See the Notes Section in
fetch_openml's API doc for details.
  warn(
```

[32]:
```
X, y = mnist.data, mnist.target
```

[33]:
```
y
```

[33]:
```
array(['5', '0', '4', …, '4', '5', '6'], dtype=object)
```

Split Data

[34]:
```
X_train, X_test, y_train, y_test = train_test_split (X ,y , test_size=0.2,␣
↪random_state= 42)
```

[35]:
```
y_train_3 = (y_train == '3')
y_test_3 = (y_test == '3')
```

0.1.2 SGDClassifier

[36]:
```
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_3);
y_pred_3 = sgd_clf.predict(X_test)
```

Evaluate using cross-validation

[37]:
```
cross_val_score(sgd_clf, X_train, y_train_3, cv=3, scoring="accuracy")
```

[37]:
```
array([0.96930412, 0.96228639, 0.9660345 ])
```

Evaluate using confusion_matrix

[38]:
```
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_3, cv =3)
cm = confusion_matrix(y_train_3,y_train_pred)
```

[39]:
```
cm
```

[39]:
```
array([[49726,   566],
       [ 1345,  4363]], dtype=int64)
```

Evaluate using Precision, Recall, F1

[40]:
```
precision_score(y_train_3, y_train_pred), recall_score(y_train_3,y_train_pred)
```

[40]:
```
(0.8851694055589369, 0.7643658023826209)
```

[41]:
```
from sklearn.metrics import f1_score
f1_score(y_train_3, y_train_pred)
```

[41]:
```
0.8203440819780013
```

Evaluate using accuracy score

[42]:
```
accuracy_score(y_test_3, y_pred_3)
```

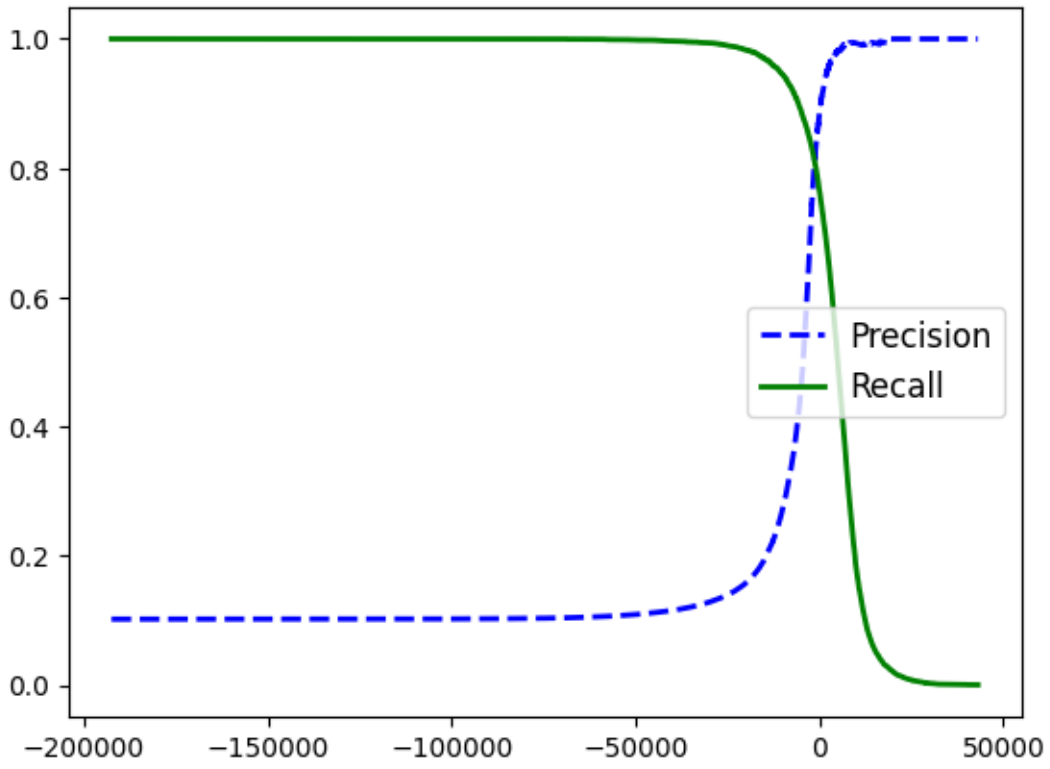
[42]:
```
0.9382142857142857
```

[43]:
```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_3, cv =3, method =␣
↪'decision_function')
```

[44]:
```
precisions, recalls, thresholds = precision_recall_curve(y_train_3,y_scores)
```

Plot Precision and Recall

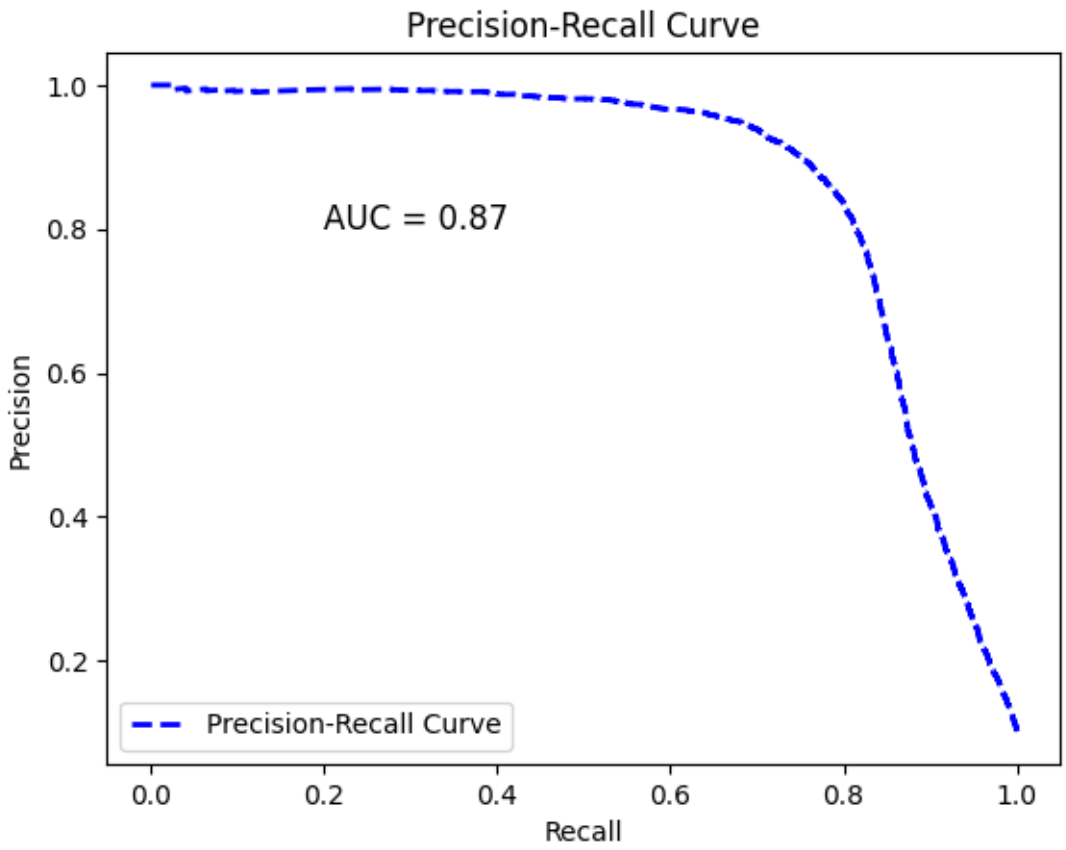
[45]:
```
plt.plot(thresholds, precisions[:-1], 'b--',label='Precision', linewidth = 2);
plt.plot(thresholds, recalls[:-1], 'g-', label = 'Recall', linewidth = 2);
plt.legend(loc = 'center right', fontsize = 12);
```



[46]:
```
auc_score = auc(recalls, precisions)

plt.plot(recalls, precisions, 'b--', label='Precision-Recall Curve',␣
 ↪linewidth=2)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')

plt.text(0.2, 0.8, f'AUC = {auc_score:.2f}', fontsize=12)
plt.show()
```



0.1.3 Random Forest

[47]:
```
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train_3);
y_pred_3 = rf_clf.predict(X_test)
```

Evaluate using Cross-validation Score

[48]:
```
cross_val_score(rf_clf, X_train, y_train_3, cv=3, scoring="accuracy")
```

[48]:
```
array([0.98226817, 0.98291102, 0.98339226])
```

Evaluate using Confusion Matrix

[49]:
```
y_train_pred = cross_val_predict(rf_clf, X_train, y_train_3, cv =3)
cm = confusion_matrix(y_train_3,y_train_pred)
cm
```

[49]:
```
array([[50262,    30],
       [  930,  4778]], dtype=int64)
```

Evaluate using Recall, Precision and F1

[50]:
```
precision_score(y_train_3, y_train_pred), recall_score(y_train_3,y_train_pred)
```

[50]:
```
(0.9937603993344426, 0.8370707778556412)
```

[51]:
```
f1_score(y_train_3, y_train_pred)
```

[51]:
```
0.9087105363255991
```

Evaluate using accuracy score

[52]:
```
accuracy_score(y_test_3, y_pred_3)
```

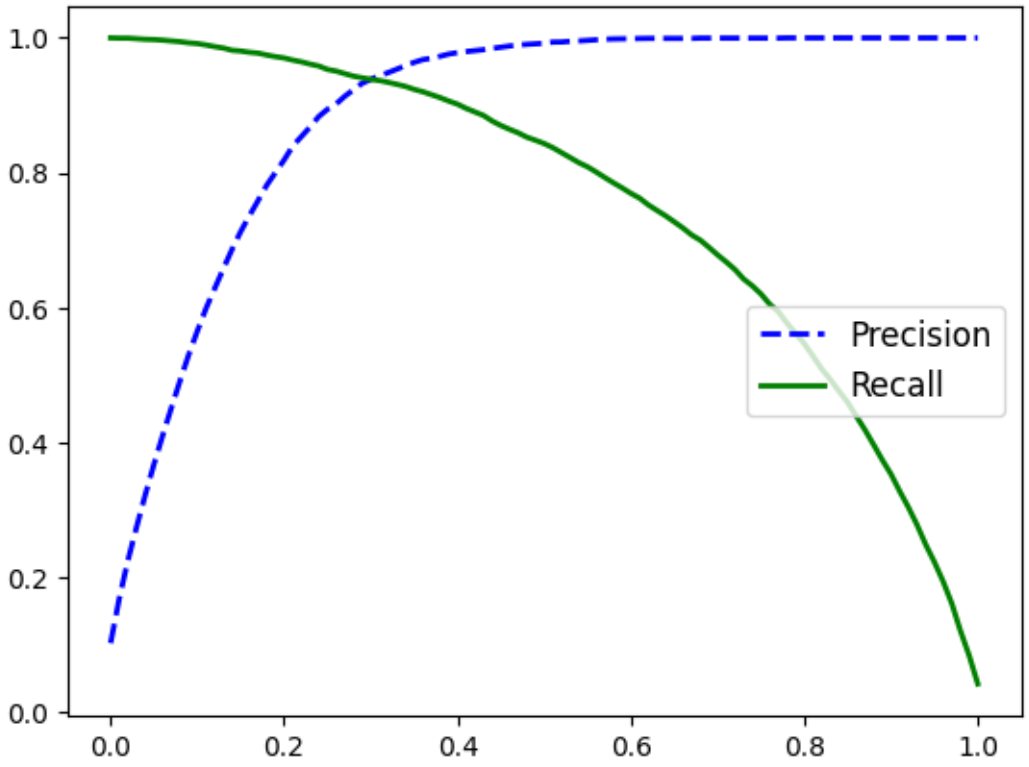
[52]:
```
0.985
```

Plot Precision and Recall

[53]:
```
y_scores = cross_val_predict(rf_clf, X_train, y_train_3, cv =3, method =␣
↪'predict_proba')[:,1]
```

[54]:
```
precisions, recalls, thresholds = precision_recall_curve(y_train_3,y_scores)
```

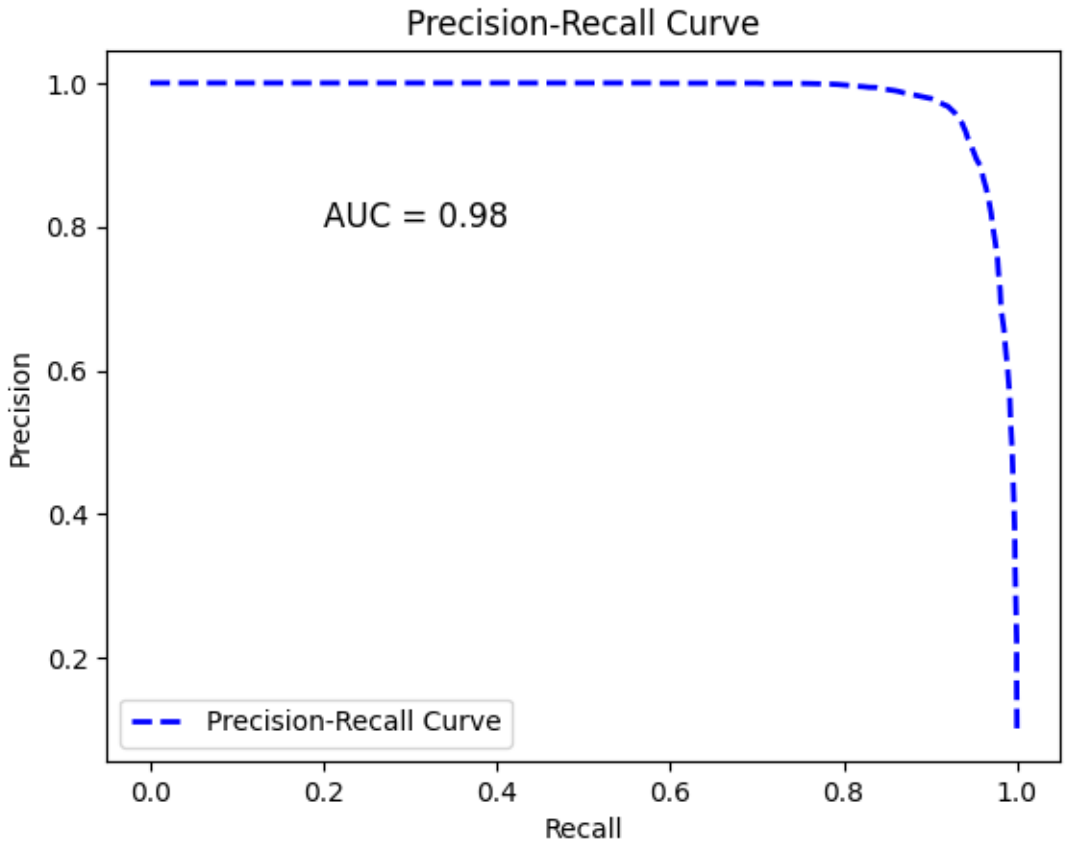
[55]:
```
plt.plot(thresholds, precisions[:-1], 'b--',label='Precision', linewidth = 2);
plt.plot(thresholds, recalls[:-1], 'g-', label = 'Recall', linewidth = 2);
plt.legend(loc = 'center right', fontsize = 12);
```



[56]:
```
auc_score = auc(recalls, precisions)


plt.plot(recalls, precisions, 'b--', label='Precision-Recall Curve',␣
 ↪linewidth=2)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')

plt.text(0.2, 0.8, f'AUC = {auc_score:.2f}', fontsize=12)
plt.show()
```



0.1.4 Support Vector Classifier

[57]:
```
svc_clf = SVC(random_state=42)
svc_clf.fit(X_train, y_train_3);
y_pred_3 = svc_clf.predict(X_test)
```

Evaluate using Cross-validation Score

[58]:
```
cross_val_score(svc_clf, X_train, y_train_3, cv=3, scoring="accuracy")
```

[58]:
```
array([0.99239299, 0.99250013, 0.99341048])
```

Evaluate using Confusion Matrix

[59]:
```
y_train_pred = cross_val_predict(svc_clf, X_train, y_train_3, cv =3)
cm = confusion_matrix(y_train_3,y_train_pred)
cm
```

[59]:
```
array([[50225,    67],
       [  338,  5370]], dtype=int64)
```

Evaluate using Recall, Precision and F1

[60]:
```
precision_score(y_train_3, y_train_pred), recall_score(y_train_3,y_train_pred)
```

[60]:
```
(0.9876770277726687, 0.9407848633496847)
```

[61]:
```
f1_score(y_train_3, y_train_pred)
```

[61]:
```
0.9636608344549125
```

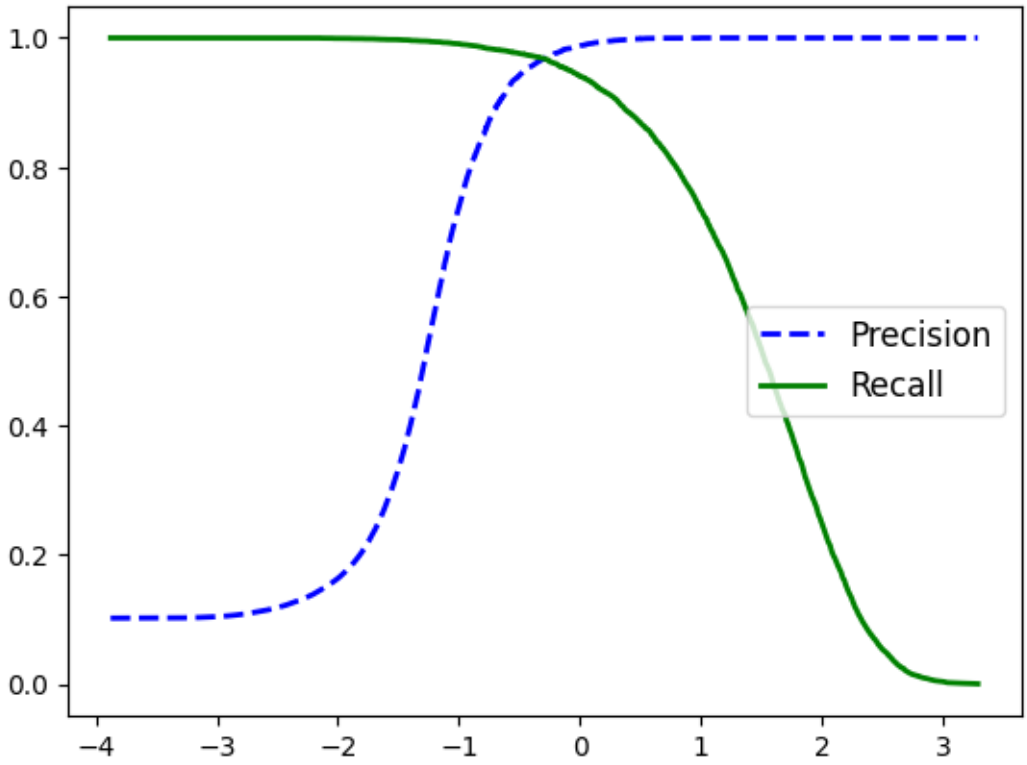
Evaluate using accuracy score

[62]:
```
accuracy_score(y_test_3, y_pred_3)
```

[62]:
```
0.9932857142857143
```

[63]:
```
y_scores = cross_val_predict(svc_clf, X_train, y_train_3, cv=3,␣
  ↪method='decision_function')
precisions, recalls, thresholds = precision_recall_curve(y_train_3,y_scores)
```

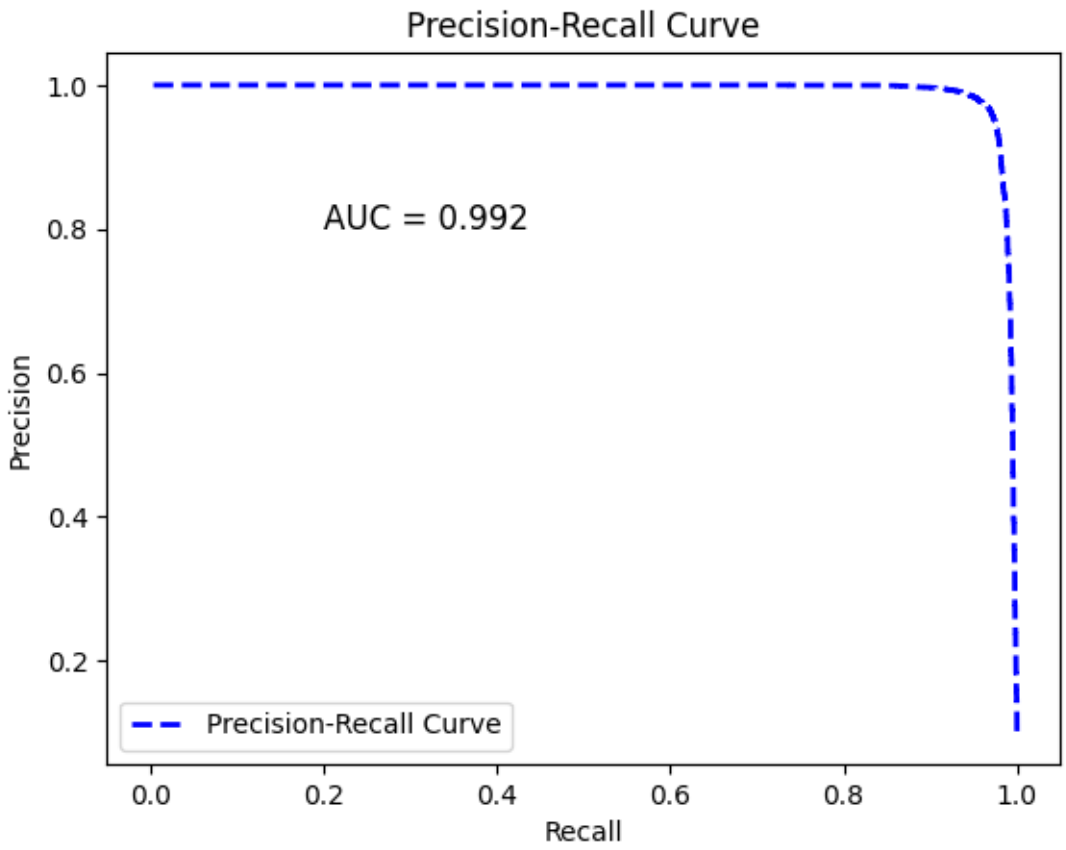
[64]:
```
plt.plot(thresholds, precisions[:-1], 'b--',label='Precision', linewidth = 2);
plt.plot(thresholds, recalls[:-1], 'g-', label = 'Recall', linewidth = 2);
plt.legend(loc = 'center right', fontsize = 12);
```



[65]:
```
auc_score = auc(recalls, precisions)
```

[66]:
```
plt.plot(recalls, precisions, 'b--', label='Precision-Recall Curve',␣
↪linewidth=2)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')

# Plot the AUC score
plt.text(0.2, 0.8, f'AUC = {auc_score:.3f}', fontsize=12)
plt.show()
```


Precision-Recall Curve
AUC = 0.992
Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0

0.2 THE END