Assignment 4

Consider a DummyBinaryClassifier that returns a random label in {True, False} for any test input that's fed to it. This classifier does not require any training! Hope, that was already obvious to you. Implement this DummyBinaryClassifier as a Python class by extending the BaseEstimator class of sklearn, so that you have mandatory methods such as fit(X, y) and predict(X) are implemented. As your guess, the fit() method would be a dummy 'pass', but the predict() method would return True or False randomly

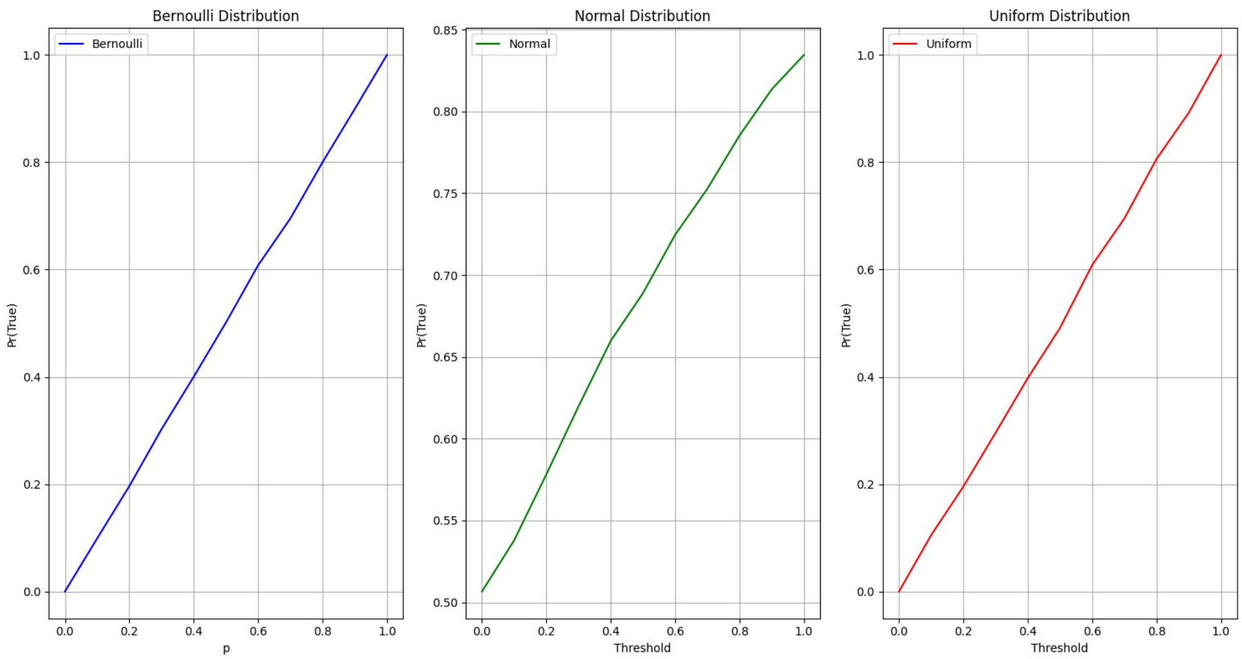
Task 1

Let's measure the label distribution (prior probability) of the predictions made by DummyBinaryClassifier. As you guessed, the label distribution is dependent on the random generator, which typically could be one of {Normal, Bernoulli or Uniform} distributions. As a part of Task 1, you are to implement all the above three generators (using libraries). You may choose the generator type while instantiating the classifier object. Moreover, Bernoulli requires 'p' as a parameter representing the probability of "True". Likewise, the normal and uniform distributions require a threshold to convert the discrete samples into Booleans. You may assume that the threshold is in [0,1] range. Typically, you will instantiate as DummyBinaryClassifier(method='bernoulli', p=0.5). The expectation is a line-plot with the x-axis represent the p in [0,1] in steps of 0.1 and the y-axis representing the Pr(True). Your plot will have 3 such lines representing 3 different random generators

```python
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin
import matplotlib.pyplot as plt

class DummyBinaryClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, method='bernoulli', p=0.5, threshold=0.5):
        self.method = method
        self.p = p
        self.threshold = threshold

    def fit(self, X, y=None):
        pass

    def predict(self, X):
        n_samples = len(X)
        if self.method == 'bernoulli':
            return np.random.rand(n_samples) < self.p
        elif self.method == 'normal':
            return np.random.normal(size=n_samples) < self.threshold
        elif self.method == 'uniform':
            return np.random.uniform(size=n_samples) < self.threshold
        else:
```

```
            raise ValueError("Method must be 'bernoulli', 'normal', or
'uniform'")


p_values = np.arange(0, 1.1, 0.1)

bernoulli_probs = []
normal_probs = []
uniform_probs = []

for p in p_values:
    # Bernoulli Distribution
    clf_bernoulli = DummyBinaryClassifier(method='bernoulli', p=p)
    predictions = clf_bernoulli.predict(np.zeros(10000))
    bernoulli_probs.append(np.mean(predictions))

    # Normal Distribution
    clf_normal = DummyBinaryClassifier(method='normal', threshold=p)
    predictions = clf_normal.predict(np.zeros(10000))
    normal_probs.append(np.mean(predictions))

    # Uniform Distribution
    clf_uniform = DummyBinaryClassifier(method='uniform', threshold=p)
    predictions = clf_uniform.predict(np.zeros(10000))
    uniform_probs.append(np.mean(predictions))

plt.figure(figsize=(15, 8))

# Subplot 1: Bernoulli Distribution
plt.subplot(1,3, 1)
plt.plot(p_values, bernoulli_probs, label='Bernoulli', color='blue')
plt.xlabel('p')
plt.ylabel('Pr(True)')
plt.title('Bernoulli Distribution')
plt.grid(True)
plt.legend()

# Subplot 2: Normal Distribution
plt.subplot(1,3, 2)
plt.plot(p_values, normal_probs, label='Normal', color='green')
plt.xlabel('Threshold')
plt.ylabel('Pr(True)')
plt.title('Normal Distribution')
plt.grid(True)
plt.legend()

# Subplot 3: Uniform Distribution
plt.subplot(1,3, 3)
plt.plot(p_values, uniform_probs, label='Uniform', color='red')
plt.xlabel('Threshold')
```

```
plt.ylabel('Pr(True)')
plt.title('Uniform Distribution')
plt.grid(True)
plt.legend()

# Adjust layout
plt.tight_layout()
plt.show()
```



Task 2

Consider the IRIS dataset, but convert the 3-class dataset into a binary class dataset by choosing the majority class as say class True and the remaining two classes as class False. Now, using the bernoulli version of the DummyBinaryClassifier, make the prediction of binary IRIS dataset.

1. Report the label prior of the binary IRIS dataset.
2. Compute the Precision, Recall, F1 of the prediction at different choice of p-values in [0,1] in steps of 0.1 and plot the P, R, C as line plots.
3. Using the P & R values, plot PRC.
4. Using TPR and FPR, plot RoC.
5. Report the AUPRC and AURoC

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.metrics import
precision_recall_curve,auc,roc_auc_score,precision_recall_fscore_suppo
```

```
rt,roc_curve
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.base import BaseEstimator, ClassifierMixin

iris=load_iris()
iris
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
        [4.7, 3.2, 1.6, 0.2],
        [4.8, 3.1, 1.6, 0.2],
        [5.4, 3.4, 1.5, 0.4],
        [5.2, 4.1, 1.5, 0.1],
        [5.5, 4.2, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.2],
        [5. , 3.2, 1.2, 0.2],
        [5.5, 3.5, 1.3, 0.2],
        [4.9, 3.6, 1.4, 0.1],
        [4.4, 3. , 1.3, 0.2],
        [5.1, 3.4, 1.5, 0.2],
        [5. , 3.5, 1.3, 0.3],
        [4.5, 2.3, 1.3, 0.3],
```

```
            [4.4, 3.2, 1.3, 0.2],
            [5. , 3.5, 1.6, 0.6],
            [5.1, 3.8, 1.9, 0.4],
            [4.8, 3. , 1.4, 0.3],
            [5.1, 3.8, 1.6, 0.2],
            [4.6, 3.2, 1.4, 0.2],
            [5.3, 3.7, 1.5, 0.2],
            [5. , 3.3, 1.4, 0.2],
            [7. , 3.2, 4.7, 1.4],
            [6.4, 3.2, 4.5, 1.5],
            [6.9, 3.1, 4.9, 1.5],
            [5.5, 2.3, 4. , 1.3],
            [6.5, 2.8, 4.6, 1.5],
            [5.7, 2.8, 4.5, 1.3],
            [6.3, 3.3, 4.7, 1.6],
            [4.9, 2.4, 3.3, 1. ],
            [6.6, 2.9, 4.6, 1.3],
            [5.2, 2.7, 3.9, 1.4],
            [5. , 2. , 3.5, 1. ],
            [5.9, 3. , 4.2, 1.5],
            [6. , 2.2, 4. , 1. ],
            [6.1, 2.9, 4.7, 1.4],
            [5.6, 2.9, 3.6, 1.3],
            [6.7, 3.1, 4.4, 1.4],
            [5.6, 3. , 4.5, 1.5],
            [5.8, 2.7, 4.1, 1. ],
            [6.2, 2.2, 4.5, 1.5],
            [5.6, 2.5, 3.9, 1.1],
            [5.9, 3.2, 4.8, 1.8],
            [6.1, 2.8, 4. , 1.3],
            [6.3, 2.5, 4.9, 1.5],
            [6.1, 2.8, 4.7, 1.2],
            [6.4, 2.9, 4.3, 1.3],
            [6.6, 3. , 4.4, 1.4],
            [6.8, 2.8, 4.8, 1.4],
            [6.7, 3. , 5. , 1.7],
            [6. , 2.9, 4.5, 1.5],
            [5.7, 2.6, 3.5, 1. ],
            [5.5, 2.4, 3.8, 1.1],
            [5.5, 2.4, 3.7, 1. ],
            [5.8, 2.7, 3.9, 1.2],
            [6. , 2.7, 5.1, 1.6],
            [5.4, 3. , 4.5, 1.5],
            [6. , 3.4, 4.5, 1.6],
            [6.7, 3.1, 4.7, 1.5],
            [6.3, 2.3, 4.4, 1.3],
            [5.6, 3. , 4.1, 1.3],
            [5.5, 2.5, 4. , 1.3],
            [5.5, 2.6, 4.4, 1.2],
```

```
       [6.1, 3. , 4.6, 1.4],
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3. , 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3. , 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3. , 5.8, 2.2],
       [7.6, 3. , 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
```

```
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'],
dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\
n--------------------\n\n**Data Set Characteristics:**\n\n    :Number
of Instances: 150 (50 in each of three classes)\n    :Number of
Attributes: 4 numeric, predictive attributes and the class\
n    :Attribute Information:\n        - sepal length in cm\n        -
sepal width in cm\n        - petal length in cm\n        - petal width
in cm\n        - class:\n                - Iris-Setosa\n
- Iris-Versicolour\n                - Iris-Virginica\
n                \n    :Summary Statistics:\n\n    ============== ====
==== ======= ===== ====================\n                    Min  Max
Mean    SD   Class Correlation\n    ============== ==== ==== =======
===== ====================\n    sepal length:   4.3  7.9   5.84   0.83
0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal
width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n
============== ==== ==== ======= ===== ====================\n\
n    :Missing Attribute Values: None\n    :Class Distribution: 33.3%
for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael
Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe
famous Iris database, first used by Sir R.A. Fisher. The dataset is
taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not
as in the UCI\nMachine Learning Repository, which has two wrong data
points.\n\nThis is perhaps the best known database to be found in the\
```

```
npattern recognition literature.  Fisher\'s paper is a classic in the
field and\nis referenced frequently to this day.  (See Duda & Hart,
for example.)  The\ndata set contains 3 classes of 50 instances each,
where each class refers to a\ntype of iris plant.  One class is
linearly separable from the other 2; the\nlatter are NOT linearly
separable from each other.\n\n|details-start|\n**References**\n|
details-split|\n\n- Fisher, R.A. "The use of multiple measurements in
taxonomic problems"\n  Annual Eugenics, 7, Part II, 179-188 (1936);
also in "Contributions to\n  Mathematical Statistics" (John Wiley, NY,
1950).\n- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and
Scene Analysis.\n  (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.
See page 218.\n- Dasarathy, B.V. (1980) "Nosing Around the
Neighborhood: A New System\n  Structure and Classification Rule for
Recognition in Partially Exposed\n  Environments".  IEEE Transactions
on Pattern Analysis and Machine\n  Intelligence, Vol. PAMI-2, No. 1,
67-71.\n- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".
IEEE Transactions\n  on Information Theory, May 1972, 431-433.\n- See
also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n
conceptual clustering system finds 3 classes in the data.\n- Many,
many more ...\n\n|details-end|',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

```
import pandas as pd
x=iris.data
y=iris.target
df=pd.DataFrame(data=x[:,0:4],columns=['SepalLengthCm','SepalWidthCm',
'PetalLengthCm','PetalWidthCm'])
```

```
df
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 150,\n  \"fields\": [\
n    {\n      \"column\": \"SepalLengthCm\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 0.828066127977863,\n
\"min\": 4.3,\n        \"max\": 7.9,\n        \"num_unique_values\":
35,\n        \"samples\": [\n          6.2,\n          4.5,\n
5.6\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SepalWidthCm\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.4358662849366982,\n        \"min\":
2.0,\n        \"max\": 4.4,\n        \"num_unique_values\": 23,\n
\"samples\": [\n          2.3,\n          4.0,\n          3.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"PetalLengthCm\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.7652982332594662,\n        \"min\": 1.0,\n        \"max\": 6.9,\n
```

```
\"num_unique_values\": 43,\n        \"samples\": [\n          6.7,\n
3.8,\n          3.7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PetalWidthCm\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.7622376689603465,\n        \"min\":
0.1,\n        \"max\": 2.5,\n        \"num_unique_values\": 22,\n
\"samples\": [\n          0.2,\n          1.2,\n          1.3\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```

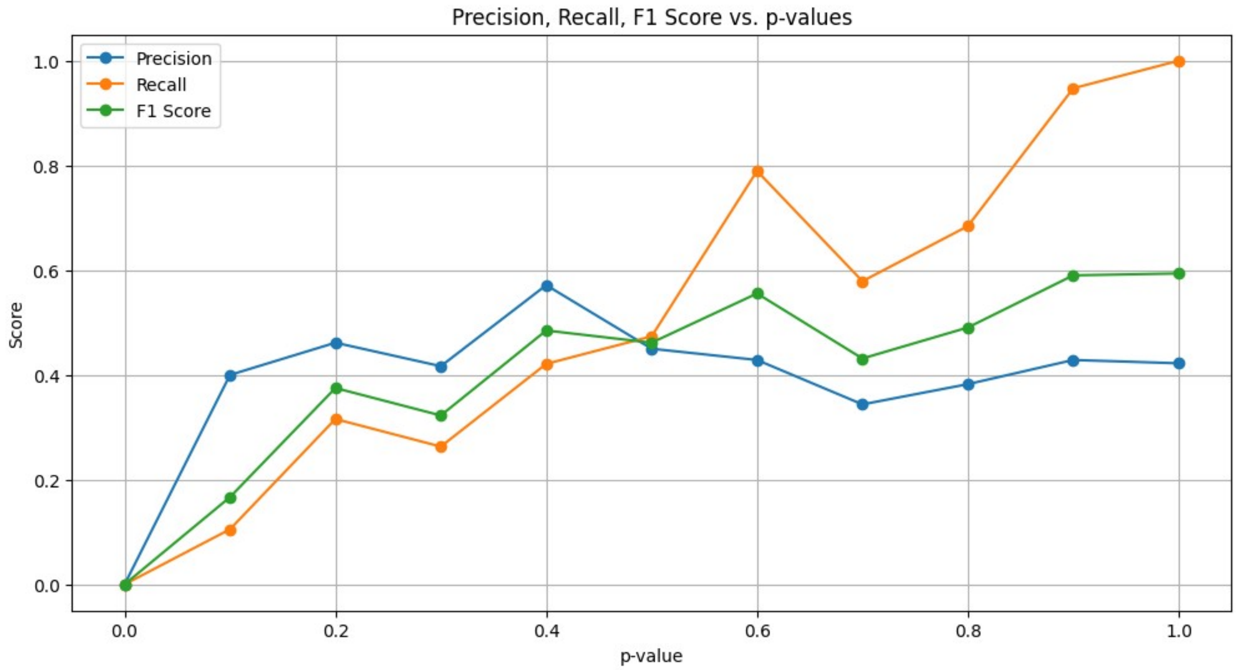

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.metrics import precision_recall_fscore_support,
roc_curve, auc, precision_recall_curve
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.base import BaseEstimator, ClassifierMixin

iris = load_iris()
X = iris.data
y = iris.target

y_binary = (y == 0).astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y_binary,
test_size=0.3, random_state=42)

class DummyBinaryClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, method='bernoulli', p=0.5, threshold=0.5):
        self.method = method
        self.p = p

    def fit(self, X, y=None):
        pass

    def predict(self, X):
        n_samples = len(X)
        return np.random.rand(n_samples) < self.p

# 1. Report the label prior of the binary IRIS dataset
label_prior = np.mean(y_binary)
print(f"Label Prior (Pr(True)) of Binary IRIS Dataset:
{label_prior:.2f}")
print()

precision_scores = []
recall_scores = []
f1_scores = []
auprc_scores = []
```

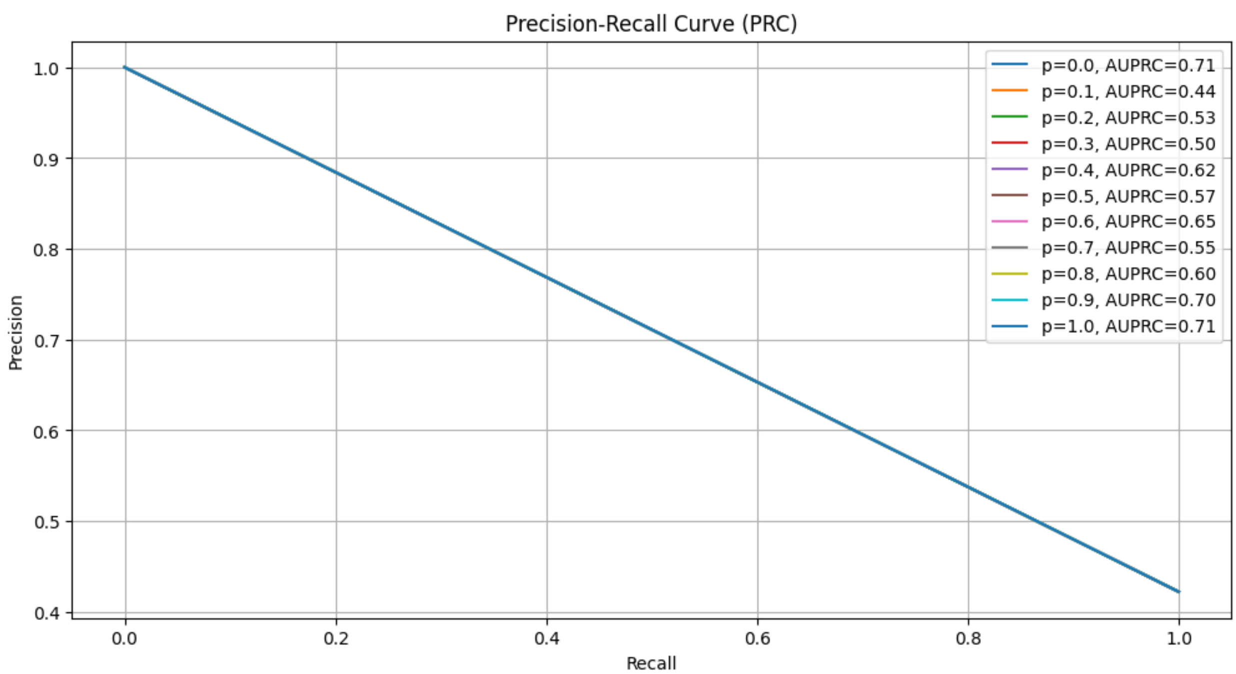
```
aurocs = []
tprs = []
fprs = []

# 2. Compute Precision, Recall, F1 for different p-values in [0,1]
p_values = np.arange(0, 1.1, 0.1)
for p in p_values:
    clf = DummyBinaryClassifier(method='bernoulli', p=p)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    # Compute Precision, Recall, F1
    precision, recall, f1, _ = precision_recall_fscore_support(y_test,
y_pred, average='binary')
    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)

    # 3. Precision-Recall Curve
    precision_curve, recall_curve, _ = precision_recall_curve(y_test,
y_pred)
    auprc = auc(recall_curve, precision_curve)
    auprc_scores.append(auprc)

    # 4. ROC Curve
    fpr, tpr, _ = roc_curve(y_test, y_pred)
    roc_auc = auc(fpr, tpr)
    aurocs.append(roc_auc)
    tprs.append(tpr)
    fprs.append(fpr)

# Plot Precision, Recall, F1 vs. p-values
plt.figure(figsize=(12, 6))
plt.plot(p_values, precision_scores, label='Precision', marker='o')
plt.plot(p_values, recall_scores, label='Recall', marker='o')
plt.plot(p_values, f1_scores, label='F1 Score', marker='o')
plt.xlabel('p-value')
plt.ylabel('Score')
plt.title('Precision, Recall, F1 Score vs. p-values')
plt.legend()
plt.grid(True)
plt.show()

# 5. Plot Precision-Recall Curve
plt.figure(figsize=(12, 6))
for i, p in enumerate(p_values):
    plt.plot(recall_curve, precision_curve, label=f'p={p:.1f},
AUPRC={auprc_scores[i]:.2f}')
plt.xlabel('Recall')
plt.ylabel('Precision')
```

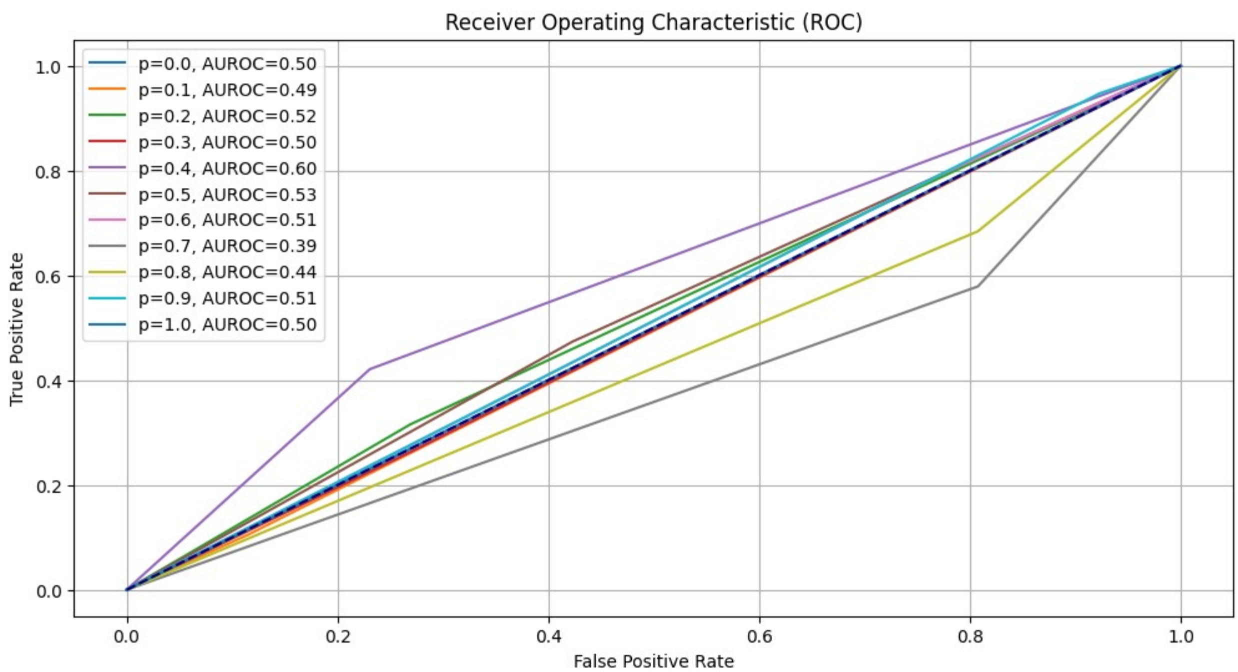
```python
plt.title('Precision-Recall Curve (PRC)')
plt.legend()
plt.grid(True)
plt.show()

# Plot ROC Curve
plt.figure(figsize=(12, 6))
for i, p in enumerate(p_values):
    plt.plot(fprs[i], tprs[i], label=f'p={p:.1f},
AUROC={aurocs[i]:.2f}')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend()
plt.grid(True)
plt.show()

# Report AUPRC and AUROC
print(f"Average AUPRC across p-values: {np.mean(auprc_scores):.2f}")
print(f"Average AUROC across p-values: {np.mean(aurocs):.2f}")
```

```
Label Prior (Pr(True)) of Binary IRIS Dataset: 0.33
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1471: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```


Precision, Recall, F1 Score vs. p-values
Precision
Recall
F1 Score
Score
p-value
0.0
0.2
0.4
0.6
0.8
1.0


Precision-Recall Curve (PRC)
p=0.0, AUPRC=0.71
p=0.1, AUPRC=0.44
p=0.2, AUPRC=0.53
p=0.3, AUPRC=0.50
p=0.4, AUPRC=0.62
p=0.5, AUPRC=0.57
p=0.6, AUPRC=0.65
p=0.7, AUPRC=0.55
p=0.8, AUPRC=0.60
p=0.9, AUPRC=0.70
p=1.0, AUPRC=0.71
Precision
Recall



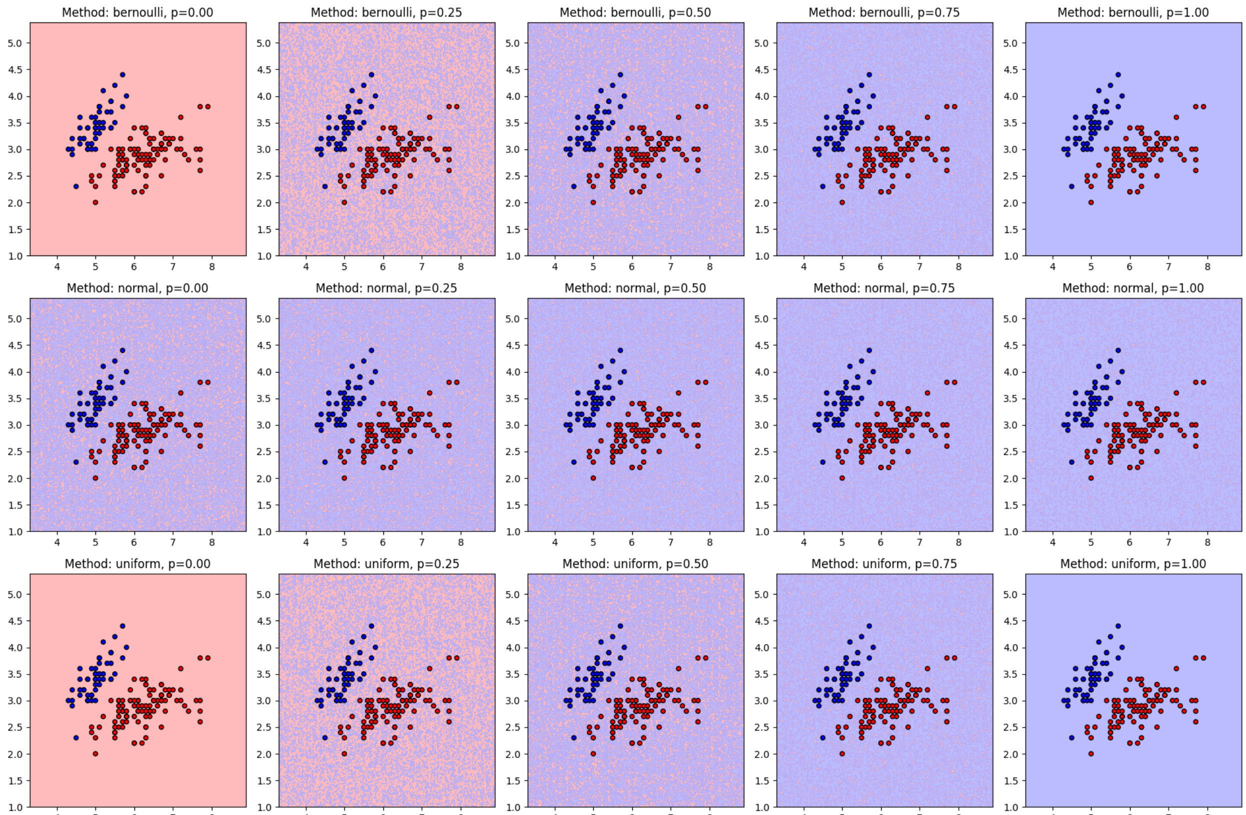

```
Average AUPRC across p-values: 0.60
Average AUROC across p-values: 0.50
```

Task 3

Generate the visualization of the decision boundaries induced by DummyBinaryClassifier at different values of p in [0, 1] in steps of 0.25 for all the three random generators.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.base import BaseEstimator, ClassifierMixin
from matplotlib.colors import ListedColormap

class DummyBinaryClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, method='bernoulli', p=0.5, threshold=0.5):
        self.method = method
        self.p = p
        self.threshold = threshold

    def fit(self, X, y=None):
        pass

    def predict(self, X):
        n_samples = len(X)
        if self.method == 'bernoulli':
            return np.random.rand(n_samples) < self.p
```

```
        elif self.method == 'normal':
            return np.random.randn(n_samples) < self.threshold
        elif self.method == 'uniform':
            return np.random.uniform(0, 1, n_samples) < self.threshold

iris = load_iris()
X = iris.data[:, :2]
y = (iris.target == 0).astype(int)

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

p_values = np.arange(0, 1.25, 0.25)
methods = ['bernoulli', 'normal', 'uniform']

fig, axes = plt.subplots(len(methods), len(p_values), figsize=(18,
12))
cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
cmap_bold = ['#FF0000', '#0000FF']

for i, method in enumerate(methods):
    for j, p in enumerate(p_values):
        clf = DummyBinaryClassifier(method=method, p=p, threshold=p)
        clf.fit(X, y)

        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)

        ax = axes[i, j]
        ax.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.8)
        ax.scatter(X[:, 0], X[:, 1], c=y,
cmap=ListedColormap(cmap_bold), edgecolor='k', s=20)
        ax.set_xlim(xx.min(), xx.max())
        ax.set_ylim(yy.min(), yy.max())
        ax.set_title(f'Method: {method}, p={p:.2f}')


plt.tight_layout()
plt.show()
```


Method: bernoulli, p=0.00
Method: bernoulli, p=0.25
Method: bernoulli, p=0.50
Method: bernoulli, p=0.75
Method: bernoulli, p=1.00
Method: normal, p=0.00
Method: normal, p=0.25
Method: normal, p=0.50
Method: normal, p=0.75
Method: normal, p=1.00
Method: uniform, p=0.00
Method: uniform, p=0.25
Method: uniform, p=0.50
Method: uniform, p=0.75
Method: uniform, p=1.00