Tree-based (Ensemble) Models and Handling Imbalanced Data [30 points]

For this problem, we will use the wine quality dataset on which the task is a binary classification of whether a given wine is of low or high quality based on different physicochemical features.

The dataset consists of a set of physicochemical features as inputs and the target is wine quality stored in the target column, where a value of 1 corresponds to an instance of high quality wine and -1 corresponds to an instance of low quality ones.

Loading the data (3pts)

Load the data from library.

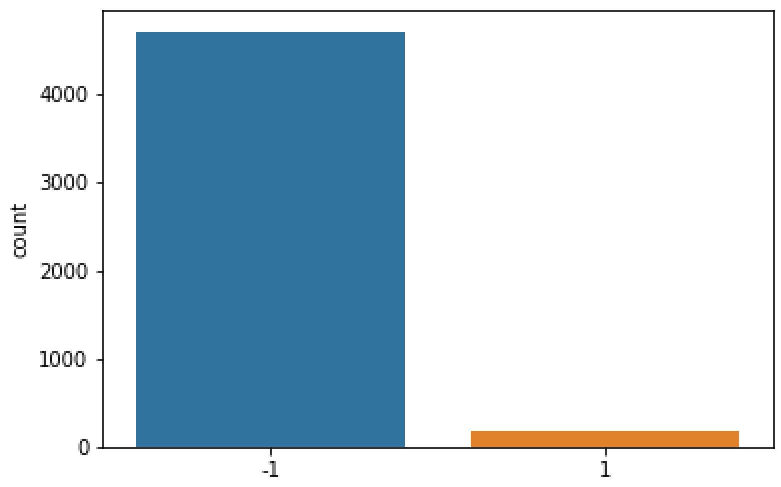
In [1]:
```
# Use wine dataset from imlearn
from imblearn.datasets import fetch_datasets
import numpy as np
import pandas as pd

datasets = fetch_datasets()

# Wine quality dataset contains 12 features, descriptions found here:
# https://archive.ics.uci.edu/ml/datasets/wine+quality
data = datasets['wine_quality']

X, y = data.data, data.target

# Use X_train, X_test, y_train, y_test for all of the following questions
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

Question: Draw the class distribution of the dataset. What are possible problems if we train a classification model directly on this dataset?

In [10]:
```
# import libraries for plotting class distribution
import matplotlib.pyplot as plt
import seaborn as sns

high_quality_ratio = round((y == 1).sum()/(y == -1).sum() * 100, 2)
print('Percentage of high quality observations: {}%'.format(high_quality_ratio))

# color coding for 2 classes
colors = ["#0101DF", "#DF0101"]

## code to plot the class distribution. Hint: countplot in seaborn
sns.countplot(y)
```

```
Percentage of high quality observations: 3.88%
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bc541dcc18>
```



Training and testing a Random Forest classfier directly on the data (3pts)

Let's first train a random forest classifier with default parameters using X_train and y_train and test the performance on the test data.

In [11]:
```
from sklearn.ensemble import RandomForestClassifier # class for random forest clas
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score

## Instantiate and fit a random forest classifier to the training data
rf_classifier = RandomForestClassifier(n_estimators=10)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)

## Measure and print out the accuracy and recall
test_acc = accuracy_score(y_pred, y_test)

print('Random forest classifier accuracy:', test_acc)

test_recall = recall_score(y_pred, y_test)

print('Random forest classifier recall:', test_recall)
```

```
Random forest classifier accuracy: 0.950204081632653
Random forest classifier recall: 0.7272727272727273
```

Quetion: Compute the recall and accuracy scores of the random forest classifier. How is the gap between the accuracy and recall scores? Provide an explanation.

There is a gap of about 23%. Accuracy measures the error the classification makes, and a high accuracy rate in this case makes sense since we have a small percentage of high quality wine; most of the predictions will be correct. Recall measures the proportion of high quality wines that were correctly classified, so this random forest did a fairly good job of calling true positives.

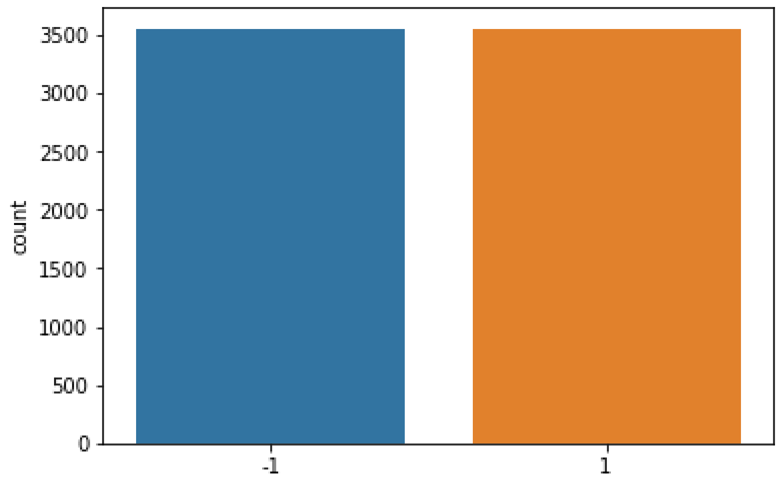
Data balancing via Smote (6pts)

In [4]:
```
from imblearn.over_sampling import SMOTE #Over sampling
import numpy as np

## Instantiate smote and balance training data only
sm = SMOTE(random_state=0)
X_resampled, y_resampled = sm.fit_resample(X_train, y_train)

## Compute and print percentage of high quality wine after balancing
percent_high_quality = round((y_resampled == 1).sum()/((y_resampled == -1).sum() +
print('Percentage of high quality counts in the balanced data: {}%'.format(percent
```

```
Percentage of high quality counts in the balanced data: 50.0%
```

Question: Plot the distribution of balanced training data.

In [5]:
```
colors = ["#0101DF", "#DF0101"]
## plot the class distribution of training data after balanced

ax = sns.countplot(y_resampled)
```



Let's retrain and test our random forest model on the balanced training data

In [6]:
```
## Instantiate random forest and train on balanced training data
rf_classifier_res = RandomForestClassifier(n_estimators=10)
rf_classifier_res.fit(X_resampled, y_resampled)
y_pred_res = rf_classifier_res.predict(X_test)

test_acc = accuracy_score(y_pred_res, y_test)

print('Random forest classifier accuracy:', test_acc)

test_recall = recall_score(y_pred_res, y_test)

print('Random forest classifier recall:', test_recall)
```

```
Random forest classifier accuracy: 0.9420408163265306
Random forest classifier recall: 0.4418604651162791
```

Question: Compute the recall and accuracy scores of the new random forest classifier. How do the accuracy and recall change compared to those without data balancing?

The accuracy is very similar for both classifiers, however the recall score is lower for the resampled data. This is likely because now there are more positive examples in the data, so the classifier will get more of them wrong.

Control complexity of the model (18pts)

Control the depth of decision trees in our ensemble (6pts)

By default, the decision trees in random forest are expanded until all leaves are pure or until all leaves contain less than a certain number set by min_samples_split parameter. Let's try a fixed maximum depth that the tree can expand.

In [7]:
```
## Instantiate model with max depth trees being 3
rf_classifier_controlled = RandomForestClassifier(n_estimators=10, max_depth=3)
rf_classifier_controlled.fit(X_resampled, y_resampled)
y_pred_con = rf_classifier_controlled.predict(X_test)

test_acc = accuracy_score(y_pred_con, y_test)
print('Random forest classifier accuracy:', test_acc)

test_recall = recall_score(y_test, y_pred_con)
print('Random forest classifier recall:', test_recall)
```

```
Random forest classifier accuracy: 0.8114285714285714
Random forest classifier recall: 0.5757575757575758
```

Question: Compute the recall and accuracy scores of the new random forest classifier. How do the accuracy and recall change compared to those in the default parameter case?

Accuracy has gone down by about 13%, but recall has increase by 12%, so this classifier is getting more of the true positives correct.

Choose the number of trees in the forest (6pts)

By default, we use 10 random trees. Let's increase this number to 100

In [13]:
```
## Instantiate model with max depth of 3 and 100 decision trees
rf_classifier_con2 = RandomForestClassifier(n_estimators=100, max_depth=3)
rf_classifier_con2.fit(X_resampled, y_resampled)
y_pred_con2 = rf_classifier_con2.predict(X_test)

test_acc = accuracy_score(y_pred_con2, y_test)
print('Random forest classifier accuracy:', test_acc)

test_recall = test_recall = recall_score(y_pred_con2, y_test)
print('Random forest classifier recall:', test_recall)
```

```
Random forest classifier accuracy: 0.8195918367346938
Random forest classifier recall: 0.15246636771300448
```

Question: Compute the recall and accuracy scores of the random forest classifier. How do the accuracy and recall change compared to those with 10 trees? What do the results imply about increasing the number of trees?

The accuracy is almost the same, but the recall score dropped dramatically. This implies that increasing the number of trees can lead to overfitting the data.

Tree pruning by min_impurity_decrease (6pts)

By default, the tree keeps expanding until the impurity is 0. However, we can specify a minimum impurity decrease amount under which nodes in the tree stop branching. RandomForestClassifier in sklearn use min_impurity_decrease for setting this threshold. Let's try that on our problem.

In [15]:
```
## Instantiate model with min impurity decrease of 0.001
rf_classifier_min_imp = RandomForestClassifier(n_estimators=100, max_depth=3, min_
rf_classifier_min_imp.fit(X_resampled, y_resampled)
y_pred_min_imp = rf_classifier_min_imp.predict(X_test)

test_acc = accuracy_score(y_pred_min_imp, y_test)
print('Random forest classifier accuracy:', test_acc)

test_recall = test_recall = recall_score(y_pred_min_imp, y_test)
print('Random forest classifier recall:', test_recall)
```

```
Random forest classifier accuracy: 0.8228571428571428
Random forest classifier recall: 0.16740088105726872
```

Question: Compute the recall and accuracy scores of the random forest classifier. How does the recall change compared to those with 10 trees and max_depth = 3?

The accuracy with a min_impurity set increases by .02, so we see a little improvement there, but the recall score is still quite low compared to the classifier with 10 trees and depth of 3.