Exercise-Ensemble-Classifers

March 3, 2024

1 Exercise - Ensemble

In this exercise, we will focus on underage drinking. The data set contains data about high school students. Each row represents a single student. The columns include the characteristics of deidentified students. This is a binary classification task: predict whether a student drinks alcohol or not (this is the **alc** column: 1=Yes, 0=No). This is an important prediction task to detect underage drinking and deploy intervention techniques.

1.1 Description of Variables

The description of variables are provided in “Alcohol - Data Dictionary.docx”

1.2 Goal

Use the **alcohol.csv** data set and build a model to predict **alc**.

2 Read and Prepare the Data

[ ]:
```
# Common imports

import pandas as pd
import numpy as np

import time

np.random.seed(1)

pd.set_option('display.max_colwidth', None)
```

3 Get the data

[ ]:
```
#We will predict the "price" value in the data set:

alcohol = pd.read_csv("alcohol.csv")
alcohol.head()
```

```
[ ]:    age  Medu  Fedu  traveltime  studytime  failures  famrel  freetime  goout  \
     0   18     2     1           4          2         0       5         4      2
     1   18     4     3           1          0         0       4         4      2
     2   15     4     3           2          3         0       5         3      4
     3   15     3     3           1          4         0       4         3      3
     4   17     3     2           1          2         0       5         3      5

        health  absences gender  alc
     0       5         2      M    1
     1       3         9      M    1
     2       5         0      F    0
     3       3        10      F    0
     4       5         2      M    1
```

```
[ ]: ## Identify any issues with data imbalance

     alcohol['alc'].value_counts() # we can see that these are a bit imbalanced, but␣
      ↪nothing to be too concerned about. If the imbalance was greater, use one of␣
      ↪the techniques to balance the data that we discussed in data mining.


     # If you had not seen how to address data imbalance, you would do this only on␣
      ↪the test set (so later on in this code). See the section later in this␣
      ↪document.
```

```
[ ]: alc
     0    17757
     1    16243
     Name: count, dtype: int64
```

3.1 Feature Engineering: Derive a new column

Examples: - Ratio of study time to travel time - Student is younger than 18 or not - Average of father's and mother's level of education - (etc.)

```
[ ]: alcohol['study_2_travel'] = (alcohol['studytime'] / alcohol['traveltime']).
      ↪replace([np.inf, -np.inf], np.nan)
     alcohol['younger_than_18'] = (alcohol['age'] < 18).astype(int)
     alcohol['avg_edu'] = (alcohol['Medu'] + alcohol['Fedu']) / 2

     alcohol.head()
```

```
[ ]:    age  Medu  Fedu  traveltime  studytime  failures  famrel  freetime  goout  \
     0   18     2     1           4          2         0       5         4      2
     1   18     4     3           1          0         0       4         4      2
     2   15     4     3           2          3         0       5         3      4
     3   15     3     3           1          4         0       4         3      3
```

```
4   17     3     2           1          2         0        5         3      5

   health  absences gender  alc  study_2_travel  younger_than_18  avg_edu
0       5         2      M    1             0.5                0      1.5
1       3         9      M    1             0.0                0      3.5
2       5         0      F    0             1.5                1      3.5
3       3        10      F    0             4.0                1      3.0
4       5         2      M    1             2.0                1      2.5
```

[ ]:
```python
# encode gender M and F to 1 and 0 respectively

alcohol = pd.get_dummies(alcohol, columns=['gender', 'alc'], drop_first=True,
  dtype='int')

alcohol.head()
```

[ ]:
```
   age  Medu  Fedu  traveltime  studytime  failures  famrel  freetime  goout  \
0   18     2     1           4          2         0       5         4      2
1   18     4     3           1          0         0       4         4      2
2   15     4     3           2          3         0       5         3      4
3   15     3     3           1          4         0       4         3      3
4   17     3     2           1          2         0       5         3      5

   health  absences  study_2_travel  younger_than_18  avg_edu  gender_M  alc_1
0       5         2             0.5                0      1.5         1      1
1       3         9             0.0                0      3.5         1      1
2       5         0             1.5                1      3.5         0      0
3       3        10             4.0                1      3.0         0      0
4       5         2             2.0                1      2.5         1      1
```

[ ]:
```python
alcohol = alcohol.rename(columns={'alc_1': 'alc_use'})

alcohol.head()
```

[ ]:
```
   age  Medu  Fedu  traveltime  studytime  failures  famrel  freetime  goout  \
0   18     2     1           4          2         0       5         4      2
1   18     4     3           1          0         0       4         4      2
2   15     4     3           2          3         0       5         3      4
3   15     3     3           1          4         0       4         3      3
4   17     3     2           1          2         0       5         3      5

   health  absences  study_2_travel  younger_than_18  avg_edu  gender_M  \
0       5         2             0.5                0      1.5         1
1       3         9             0.0                0      3.5         1
2       5         0             1.5                1      3.5         0
3       3        10             4.0                1      3.0         0
4       5         2             2.0                1      2.5         1
```

```
   alc_use
0        1
1        1
2        0
3        0
4        1
```

4 Data Prep

[ ]:
```python
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import FunctionTransformer
```

[ ]:
```python
# Split into X and y

y = alcohol['alc_use']
X = alcohol.drop('alc_use', axis=1)
```

4.1 Identify the numeric, binary, and categorical columns

[ ]:
```python
# Identify the numerical columns
numeric_columns = X.select_dtypes('number').columns.to_list()

# Identify the categorical columns
categorical_columns = X.select_dtypes('object').columns.to_list()
```

[ ]:
```python
numeric_columns
```

[ ]:
```
['age',
 'Medu',
 'Fedu',
 'traveltime',
 'studytime',
 'failures',
 'famrel',
 'freetime',
 'goout',
 'health',
 'absences',
 'study_2_travel',
 'younger_than_18',
```

```
 'avg_edu',
 'gender_M']
```

[ ]:
```
categorical_columns
```

[ ]:
```
[]
```

[ ]:
```
binary_columns = [col for col in X.columns if X[col].nunique() == 2]
binary_columns
```

[ ]:
```
['younger_than_18', 'gender_M']
```

[ ]:
```
for binary_col in binary_columns:
    numeric_columns.remove(binary_col)

numeric_columns
```

[ ]:
```
['age',
 'Medu',
 'Fedu',
 'traveltime',
 'studytime',
 'failures',
 'famrel',
 'freetime',
 'goout',
 'health',
 'absences',
 'study_2_travel',
 'avg_edu']
```

5 Split data (train/test)

[ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
 ↪random_state=42)
```

5.1 Address any data imbalance issues

> [!NOTE] See presentation for more details on the pros and cons of each technique. It's up to you to decide which one to use, and justify why you chose it. Another approach would be to train the models on resampled data using each of the 4 techniques and report which approach resulted in the best outcome. Also, it's important to note that you should not resample the test data, only the training data.

[ ]:
```
# There are three main techniques to balance the data (see powerpoint␣
↪presentation for more details on these techniques):
# 1. Random Over Sampling
# 2. Random Under Sampling
# 3. SMOTE (Synthetic Minority Over-sampling Technique)
# 4. ADASYN (Adaptive Synthetic Sampling)

# from imblearn.over_sampling import RandomOverSampler
# from imblearn.under_sampling import RandomUnderSampler
# from imblearn.over_sampling import SMOTE
# from imblearn.over_sampling import ADASYN

# ros = RandomOverSampler(random_state=0)
# rus = RandomUnderSampler(random_state=0)
# smote = SMOTE(random_state=0)
# adasyn = ADASYN(random_state=0)

# X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)
# X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_train)
# X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
# X_train_resampled, y_train_resampled = adasyn.fit_resample(X_train, y_train)
```

6 Pipeline

[ ]:
```
numeric_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())
    ]
)
```

[ ]:
```
categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='constant', fill_value=-1)),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ]
)
```

[ ]:
```
binary_transformer = Pipeline( steps=[
        ('imputer', SimpleImputer(strategy='most_frequent'))
    ]
)
```

[ ]:
```
preprocessor = ColumnTransformer(
    [
        ('num', numeric_transformer, numeric_columns),
```

```
        ('cat', categorical_transformer, categorical_columns), # we don't have
 ↪any categorical columns in this data set, so we don't need to include this
 ↪line
        ('binary', binary_transformer, binary_columns)
    ],
    remainder='passthrough'
)
```

7 Transform: fit_transform() for TRAIN

[ ]:
```
#Fit and transform the train data
X_train = preprocessor.fit_transform(X_train)
```

[ ]:
```
X_train.shape
```

[ ]:
```
(23800, 15)
```

8 Tranform: transform() for TEST

[ ]:
```
# Transform the test data
X_test = preprocessor.transform(X_test)

X_test
```

[ ]:
```
array([[-1.23984621,  0.33104402,  1.76705606, …,  1.01168573,
         1.        ,  0.        ],
       [-1.23984621, -0.30388608,  0.04019664, …, -0.1702172 ,
         1.        ,  1.        ],
       [-0.28670367,  0.33104402,  0.04019664, …,  0.22375045,
         1.        ,  1.        ],
       …,
       [ 0.66643886, -0.30388608,  0.04019664, …, -0.1702172 ,
         1.        ,  0.        ],
       [-1.23984621, -0.93881619,  0.04019664, …, -0.56418484,
         1.        ,  1.        ],
       [-1.23984621,  0.96597412,  0.04019664, …,  0.61771809,
         1.        ,  0.        ]])
```

[ ]:
```
X_test.shape
```

[ ]:
```
(10200, 15)
```

9 Develop Models

9.1 Create dataframe to store results

[ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_auc_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression

results = pd.DataFrame(columns=['Model', 'Duration', 'Accuracy', 'Precision',␣
↪'Recall', 'F1', 'AUC', 'Best Parameters'])

iters = 5
folds = 2
```

9.2 Train a Logistic Regress Classifier (use random search hyperparameter tuning)

[ ]:
```
start = time.time()

# setup parameters for RandomizedSearchCV for Logistic Regression
param_distributions = {
    'C': np.logspace(-4, 4, 100),
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}

log_reg = LogisticRegression()
log_reg_cv = RandomizedSearchCV(log_reg, param_distributions, n_iter=iters,␣
↪cv=folds, scoring='f1', verbose=1, n_jobs=-1, random_state=42)
log_reg_cv.fit(X_train, y_train)
model01 = log_reg_cv.best_estimator_

# calculate accuracy, precision, recall, f1, auc
y_pred = model01.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

end = time.time()

# Pandas used to have a append method, but it is now deprecated. The␣
↪recommended way is to use the concat method or the following method:
```

```
results.loc[len(results.index)] = ['Logistic Regression', end-start, accuracy,␣
↪precision, recall, f1, auc, str(log_reg_cv.best_params_)]
results
```

```
Fitting 2 folds for each of 5 candidates, totalling 10 fits
```

```
[ ]:                  Model  Duration  Accuracy  Precision    Recall        F1  \
0  Logistic Regression  0.702046  0.821373   0.822446  0.801306  0.811738

        AUC                                                  Best Parameters
0  0.820623  {'solver': 'liblinear', 'penalty': 'l2', 'C': 0.6280291441834259}
```

9.3 Train a random forest classifier (use random search hyperparameter tuning)

```
[ ]: start = time.time()

# set up parameters for RandomizedSearchCV for Random Forest

from sklearn.ensemble import RandomForestClassifier

param_distributions = {
    'n_estimators': [int(x) for x in np.linspace(start=200, stop=2000, num=10)],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [int(x) for x in np.linspace(2, 100, num=2)] + [None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

rf = RandomForestClassifier()
rf_cv = RandomizedSearchCV(rf, param_distributions, n_iter=iters, cv=folds,␣
↪scoring='f1', verbose=1, n_jobs=-1, random_state=42)
rf_cv.fit(X_train, y_train)
model02 = rf_cv.best_estimator_

# calculate accuracy, precision, recall, f1, auc
y_pred = model02.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
auc = roc_auc_score(y_test, y_pred)

end = time.time()
```

```
results.loc[len(results.index)] = ['Random Forest', end-start, accuracy,␣
↪precision, recall, f1, auc, str(rf_cv.best_params_)]

results
```

```
Fitting 2 folds for each of 5 candidates, totalling 10 fits
```

[ ]:

```
                 Model  Duration  Accuracy  Precision    Recall        F1  \
0  Logistic Regression  0.702046  0.821373   0.822446  0.801306  0.811738
1        Random Forest  8.487113  0.816667   0.812706  0.803754  0.808205

        AUC  \
0  0.820623
1  0.816184

                                     Best Parameters
0                                                             {'solver':
'liblinear', 'penalty': 'l2', 'C': 0.6280291441834259}
1  {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 4,
'max_features': 'sqrt', 'max_depth': 100, 'bootstrap': True}
```

9.4 Train an adaboost classifier (use random search hyperparameter tuning)

[ ]:

```
start = time.time()

# set up parameters for RandomizedSearchCV for adabooost

from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

param_distributions = {
    'estimator': [DecisionTreeClassifier(max_depth=1),␣
↪DecisionTreeClassifier(max_depth=2), DecisionTreeClassifier(max_depth=3),␣
↪DecisionTreeClassifier(max_depth=4)],
    'n_estimators': [50, 100, 200, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.5, 1.0]
}

ada = AdaBoostClassifier()
ada_cv = RandomizedSearchCV(ada, param_distributions, n_iter=iters, cv=folds,␣
↪scoring='f1', verbose=1, n_jobs=-1, random_state=42)
ada_cv.fit(X_train, y_train)
model03 = ada_cv.best_estimator_

# calculate accuracy, precision, recall, f1, auc
y_pred = model03.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
auc = roc_auc_score(y_test, y_pred)

end = time.time()

results.loc[len(results.index)] = ['AdaBoost', end-start, accuracy, precision,␣
 ↪recall, f1, auc, str(ada_cv.best_params_)]

results
```

```
Fitting 2 folds for each of 5 candidates, totalling 10 fits
```

[ ]:
```
                 Model  Duration  Accuracy  Precision    Recall        F1  \
0  Logistic Regression  0.702046  0.821373   0.822446  0.801306  0.811738
1        Random Forest  8.487113  0.816667   0.812706  0.803754  0.808205
2             AdaBoost  6.462828  0.826078   0.823810  0.811710  0.817715

        AUC  \
0  0.820623
1  0.816184
2  0.825541

                                     Best Parameters
0                                                                  {'solver':
'liblinear', 'penalty': 'l2', 'C': 0.6280291441834259}
1  {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 4,
'max_features': 'sqrt', 'max_depth': 100, 'bootstrap': True}
2                                       {'n_estimators': 500, 'learning_rate':
0.1, 'estimator': DecisionTreeClassifier(max_depth=2)}
```

9.5 KNN Classifier

[ ]:
```
# set up parameters for RandomizedSearchCV for KNN  (this is a slow process)

start = time.time()

from sklearn.neighbors import KNeighborsClassifier

param_distributions = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'p': [1, 2]
```

```
}

knn = KNeighborsClassifier()

knn_cv = RandomizedSearchCV(knn, param_distributions, n_iter=iters, cv=folds,␣
 ↪scoring='f1', verbose=1, n_jobs=-1, random_state=42)
knn_cv.fit(X_train, y_train)
model04 = knn_cv.best_estimator_

# calculate accuracy, precision, recall, f1, auc
y_pred = model04.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
auc = roc_auc_score(y_test, y_pred)

end = time.time()
results.loc[len(results.index)] = ['KNN', end-start, accuracy, precision,␣
 ↪recall, f1, auc, str(knn_cv.best_params_)]
results
```

```
Fitting 2 folds for each of 5 candidates, totalling 10 fits
```

[ ]:

```
                 Model  Duration  Accuracy  Precision    Recall        F1  \
0  Logistic Regression  0.702046  0.821373   0.822446  0.801306  0.811738
1        Random Forest  8.487113  0.816667   0.812706  0.803754  0.808205
2             AdaBoost  6.462828  0.826078   0.823810  0.811710  0.817715
3                  KNN  2.696353  0.809118   0.808133  0.790494  0.799216

        AUC  \
0  0.820623
1  0.816184
2  0.825541
3  0.808422

                                     Best Parameters
0                                                                   {'solver':
'liblinear', 'penalty': 'l2', 'C': 0.6280291441834259}
1  {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 4,
'max_features': 'sqrt', 'max_depth': 100, 'bootstrap': True}
2                                    {'n_estimators': 500, 'learning_rate':
0.1, 'estimator': DecisionTreeClassifier(max_depth=2)}
3                                                              {'weights':
'uniform', 'p': 2, 'n_neighbors': 11, 'algorithm': 'auto'}
```

9.6 XGBoost Classifier

```
[ ]: # set up parameters for RandomizedSearchCV for XGBClassifier

start = time.time()

from xgboost import XGBClassifier

param_distributions = {
    'n_estimators': [int(x) for x in np.linspace(start=200, stop=2000, num=10)],
    'max_depth': [int(x) for x in np.linspace(2, 100, num=2)] + [None],
    'learning_rate': [0.01, 0.05, 0.1, 0.5, 1.0],
    'subsample': [0.5, 0.75, 1.0],
    'colsample_bytree': [0.5, 0.75, 1.0],
    'gamma': [0, 1, 5],
    'reg_alpha': [0, 1, 5],
    'reg_lambda': [0, 1, 5]
}

xgb = XGBClassifier()
xgb_cv = RandomizedSearchCV(xgb, param_distributions, n_iter=iters, cv=folds,␣
 ↪scoring='f1', verbose=1, n_jobs=-1, random_state=42)
xgb_cv.fit(X_train, y_train)
model05 = xgb_cv.best_estimator_

# calculate accuracy, precision, recall, f1, auc
y_pred = model05.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
auc = roc_auc_score(y_test, y_pred)

end = time.time()

results.loc[len(results.index)] = ['XGBClassifier', end-start, accuracy,␣
 ↪precision, recall, f1, auc, str(xgb_cv.best_params_)]
results
```

```
Fitting 2 folds for each of 5 candidates, totalling 10 fits
```

[ ]:

	Model	Duration	Accuracy	Precision	Recall	F1 \
0	Logistic Regression	0.702046	0.821373	0.822446	0.801306	0.811738
1	Random Forest	8.487113	0.816667	0.812706	0.803754	0.808205
2	AdaBoost	6.462828	0.826078	0.823810	0.811710	0.817715
3	KNN	2.696353	0.809118	0.808133	0.790494	0.799216
4	XGBClassifier	3.602068	0.827059	0.824710	0.812933	0.818780

```
          AUC  \
0  0.820623
1  0.816184
2  0.825541
3  0.808422
4  0.826531

                                                            Best Parameters
0
{'solver': 'liblinear', 'penalty': 'l2', 'C': 0.6280291441834259}
1                           {'n_estimators': 200, 'min_samples_split': 2,
'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': 100, 'bootstrap':
True}
2                                                             {'n_estimators':
500, 'learning_rate': 0.1, 'estimator': DecisionTreeClassifier(max_depth=2)}
3
{'weights': 'uniform', 'p': 2, 'n_neighbors': 11, 'algorithm': 'auto'}
4  {'subsample': 1.0, 'reg_lambda': 0, 'reg_alpha': 0, 'n_estimators': 1000,
'max_depth': None, 'learning_rate': 0.01, 'gamma': 1, 'colsample_bytree': 0.75}
```

9.7 SVC

Uncomment the code below to train a SVC model. This model is computationally expensive and may take a long time to train.

[ ]:
```
"""

# set up parameters for RandomizedSearchCV for SVM  (this is a slow process)

start = time.time()

from sklearn.svm import SVC

param_distributions = {
    'C': [0.1, 1, 2, 5, 10, 15, 20, 40, 80, 100],
    'gamma': [1, 0.5, 0.1, 0.05, 0.01, 0.001],
    'kernel': ['rbf', 'poly', 'sigmoid']
}

svc = SVC()

svc_cv = RandomizedSearchCV(svc, param_distributions, n_iter=iters, cv=folds,␣
 ↪scoring='f1', verbose=1, n_jobs=-1, random_state=42)

svc_cv.fit(X_train, y_train)
model06 = svc_cv.best_estimator_
```

```
# calculate accuracy, precision, recall, f1, auc
y_pred = model06.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
auc = roc_auc_score(y_test, y_pred)

end = time.time()

results.loc[len(results.index)] = ['SVM', end-start, accuracy, precision,␣
↪recall, f1, auc, str(svc_cv.best_params_)]
results

"""
```

[ ]: "\n\n# set up parameters for RandomizedSearchCV for SVM (this is a slow process)\n\nstart = time.time()\n\nfrom sklearn.svm import SVC\n\nparam_distributions = {\n   'C': [0.1, 1, 2, 5, 10, 15, 20, 40, 80, 100],\n   'gamma': [1, 0.5, 0.1, 0.05, 0.01, 0.001],\n   'kernel': ['rbf', 'poly', 'sigmoid']\n}\n\nsvc = SVC()\n\nsvc_cv = RandomizedSearchCV(svc, param_distributions, n_iter=iters, cv=folds, scoring='f1', verbose=1, n_jobs=-1, random_state=42)\n\nsvc_cv.fit(X_train, y_train)\nmodel06 = svc_cv.best_estimator_\n\n# calculate accuracy, precision, recall, f1, auc\ny_pred = model06.predict(X_test)\n\naccuracy = accuracy_score(y_test, y_pred)\nprecision = precision_score(y_test, y_pred, zero_division=0)\nrecall = recall_score(y_test, y_pred, zero_division=0)\nf1 = f1_score(y_test, y_pred, zero_division=0)\nauc = roc_auc_score(y_test, y_pred)\n\nend = time.time()\n\nresults.loc[len(results.index)] = ['SVM', end-start, accuracy, precision, recall, f1, auc, str(svc_cv.best_params_)]\nresults\n\n"

9.8 Train a Voting Classifier using previous models (test both soft and hard voting)

9.8.1 Hard Voting

This is the default behavior of the VotingClassifier. In hard voting, the predicted output class is a class with the highest majority of votes i.e the class which had the highest probability of being predicted by each of the classifiers. Suppose three classifiers predicted the output class(A, A, B), so here by majority class A has been predicted.

[ ]:
```
start = time.time()

# train a voting classifier using the three models (model01, model02, model03)

from sklearn.ensemble import VotingClassifier
```

```
voting_clf = VotingClassifier(
#    estimators=[('lr', model01), ('rf', model02), ('ada', model03), ('knn',␣
 ↪model04), ('xgb', model05), ('svc', model06)],
    estimators=[('lr', model01), ('rf', model02), ('ada', model03), ('knn',␣
 ↪model04), ('xgb', model05)],
    voting='hard'
)

voting_clf.fit(X_train, y_train)

# calculate accuracy, precision, recall, f1, auc
y_pred = voting_clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
auc = roc_auc_score(y_test, y_pred)

end = time.time()

results.loc[len(results.index)] = ['Voting Classifier-Hard', end-start,␣
 ↪accuracy, precision, recall, f1, auc, '']

results
```

[ ]:

```
                    Model  Duration  Accuracy  Precision    Recall        F1  \
0     Logistic Regression  0.702046  0.821373   0.822446  0.801306  0.811738
1           Random Forest  8.487113  0.816667   0.812706  0.803754  0.808205
2                AdaBoost  6.462828  0.826078   0.823810  0.811710  0.817715
3                     KNN  2.696353  0.809118   0.808133  0.790494  0.799216
4           XGBClassifier  3.602068  0.827059   0.824710  0.812933  0.818780
5  Voting Classifier-Hard  6.679722  0.826765   0.824065  0.813137  0.818565

        AUC  \
0  0.820623
1  0.816184
2  0.825541
3  0.808422
4  0.826531
5  0.826255

                                                             Best Parameters
0
{'solver': 'liblinear', 'penalty': 'l2', 'C': 0.6280291441834259}
1                           {'n_estimators': 200, 'min_samples_split': 2,
```

```
'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': 100, 'bootstrap':
True}
2                                                               {'n_estimators':
500, 'learning_rate': 0.1, 'estimator': DecisionTreeClassifier(max_depth=2)}
3
{'weights': 'uniform', 'p': 2, 'n_neighbors': 11, 'algorithm': 'auto'}
4  {'subsample': 1.0, 'reg_lambda': 0, 'reg_alpha': 0, 'n_estimators': 1000,
'max_depth': None, 'learning_rate': 0.01, 'gamma': 1, 'colsample_bytree': 0.75}
5
```

9.8.2 Soft Voting

This voting classifier predicts the class label based on the argmax of the sums of the predicted probabilities. Soft voting takes into account the probability of each label. It predicts the class label based on the argmax of the sum of the predicted probabilities.

[ ]:
```
start = time.time()

# train a voting classifier using the three models (model01, model02, model03,␣
↪model04, model05, model06)

voting_clf = VotingClassifier(
#    estimators=[('lr', model01), ('rf', model02), ('ada', model03), ('knn',␣
↪model04), ('xgb', model05), ('svc', model06)],
    estimators=[('lr', model01), ('rf', model02), ('ada', model03), ('knn',␣
↪model04), ('xgb', model05)],
    voting='hard'
)

voting_clf.fit(X_train, y_train)

# calculate accuracy, precision, recall, f1, auc
y_pred = voting_clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
auc = roc_auc_score(y_test, y_pred)

end = time.time()

results.loc[len(results.index)] = ['Voting Classifier-Soft', end-start,␣
↪accuracy, precision, recall, f1, auc, '']

results
```

```
[ ]:                  Model  Duration  Accuracy  Precision    Recall        F1  \
0     Logistic Regression  0.702046  0.821373   0.822446  0.801306  0.811738
1           Random Forest  8.487113  0.816667   0.812706  0.803754  0.808205
2                AdaBoost  6.462828  0.826078   0.823810  0.811710  0.817715
3                     KNN  2.696353  0.809118   0.808133  0.790494  0.799216
4           XGBClassifier  3.602068  0.827059   0.824710  0.812933  0.818780
5  Voting Classifier-Hard  6.679722  0.826765   0.824065  0.813137  0.818565
6  Voting Classifier-Soft  6.676889  0.826765   0.824199  0.812933  0.818527

        AUC  \
0  0.820623
1  0.816184
2  0.825541
3  0.808422
4  0.826531
5  0.826255
6  0.826248

                                                              Best Parameters
0
{'solver': 'liblinear', 'penalty': 'l2', 'C': 0.6280291441834259}
1                           {'n_estimators': 200, 'min_samples_split': 2,
'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': 100, 'bootstrap':
True}
2                                                              {'n_estimators':
500, 'learning_rate': 0.1, 'estimator': DecisionTreeClassifier(max_depth=2)}
3
{'weights': 'uniform', 'p': 2, 'n_neighbors': 11, 'algorithm': 'auto'}
4  {'subsample': 1.0, 'reg_lambda': 0, 'reg_alpha': 0, 'n_estimators': 1000,
'max_depth': None, 'learning_rate': 0.01, 'gamma': 1, 'colsample_bytree': 0.75}
5
6
```

9.9 Train a StackedClassifier with the above models (minus the VotingClassifier)

```
[ ]: start = time.time()

# train a stacking classifier using the three models (model01, model02, model03)

from sklearn.ensemble import StackingClassifier

stacking_clf = StackingClassifier(
#    estimators=[('lr', model01), ('rf', model02), ('ada', model03), ('knn',␣
↪model04), ('xgb', model05), ('svc', model06)],
    estimators=[('lr', model01), ('rf', model02), ('ada', model03), ('knn',␣
↪model04), ('xgb', model05)],
```

```
    final_estimator=LogisticRegression()
)

stacking_clf.fit(X_train, y_train)

# calculate accuracy, precision, recall, f1, auc
y_pred = stacking_clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
auc = roc_auc_score(y_test, y_pred)

end = time.time()

results.loc[len(results.index)] = ['Stacking Classifier', end-start, accuracy,␣
 ↪precision, recall, f1, auc, '']

results
```

[ ]:

```
                      Model   Duration  Accuracy  Precision    Recall        F1  \
0       Logistic Regression   0.702046  0.821373   0.822446  0.801306  0.811738
1             Random Forest   8.487113  0.816667   0.812706  0.803754  0.808205
2                  AdaBoost   6.462828  0.826078   0.823810  0.811710  0.817715
3                       KNN   2.696353  0.809118   0.808133  0.790494  0.799216
4             XGBClassifier   3.602068  0.827059   0.824710  0.812933  0.818780
5   Voting Classifier-Hard   6.679722  0.826765   0.824065  0.813137  0.818565
6   Voting Classifier-Soft   6.676889  0.826765   0.824199  0.812933  0.818527
7       Stacking Classifier  32.927035  0.827647   0.825736  0.812933  0.819285

        AUC  \
0  0.820623
1  0.816184
2  0.825541
3  0.808422
4  0.826531
5  0.826255
6  0.826248
7  0.827097

                                                         Best Parameters
0
{'solver': 'liblinear', 'penalty': 'l2', 'C': 0.6280291441834259}
1                           {'n_estimators': 200, 'min_samples_split': 2,
'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': 100, 'bootstrap':
True}
```

```
2                                                             {'n_estimators':
500, 'learning_rate': 0.1, 'estimator': DecisionTreeClassifier(max_depth=2)}
3
{'weights': 'uniform', 'p': 2, 'n_neighbors': 11, 'algorithm': 'auto'}
4  {'subsample': 1.0, 'reg_lambda': 0, 'reg_alpha': 0, 'n_estimators': 1000,
'max_depth': None, 'learning_rate': 0.01, 'gamma': 1, 'colsample_bytree': 0.75}
5
6
7
```

9.10 Discuss the results of the models and the best model based on F1 score results.

[ ]:
```
results.to_csv('results.csv', index=False)
results
```

[ ]:
```
                     Model   Duration  Accuracy  Precision    Recall        F1  \
0      Logistic Regression   0.702046  0.821373   0.822446  0.801306  0.811738
1            Random Forest   8.487113  0.816667   0.812706  0.803754  0.808205
2                 AdaBoost   6.462828  0.826078   0.823810  0.811710  0.817715
3                      KNN   2.696353  0.809118   0.808133  0.790494  0.799216
4            XGBClassifier   3.602068  0.827059   0.824710  0.812933  0.818780
5   Voting Classifier-Hard   6.679722  0.826765   0.824065  0.813137  0.818565
6   Voting Classifier-Soft   6.676889  0.826765   0.824199  0.812933  0.818527
7      Stacking Classifier  32.927035  0.827647   0.825736  0.812933  0.819285

        AUC  \
0  0.820623
1  0.816184
2  0.825541
3  0.808422
4  0.826531
5  0.826255
6  0.826248
7  0.827097

                                                            Best Parameters
0
{'solver': 'liblinear', 'penalty': 'l2', 'C': 0.6280291441834259}
1                           {'n_estimators': 200, 'min_samples_split': 2,
'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': 100, 'bootstrap':
True}
2                                                             {'n_estimators':
500, 'learning_rate': 0.1, 'estimator': DecisionTreeClassifier(max_depth=2)}
3
{'weights': 'uniform', 'p': 2, 'n_neighbors': 11, 'algorithm': 'auto'}
4  {'subsample': 1.0, 'reg_lambda': 0, 'reg_alpha': 0, 'n_estimators': 1000,
```

```
'max_depth': None, 'learning_rate': 0.01, 'gamma': 1, 'colsample_bytree': 0.75}
5
6
7
```