산업과 예술의 만남
홍익대학교

데이터 마이닝 Term Project 보고서

과목명:	데이터 마이닝-501
담 당:	신지애 교수님
제출일:	2021 년 12월 13일
성명 :	B993007 고나영
수업시간 :	월 678 교시

1. 프로젝트 목적

데이터 마이닝에서 배운 여러 알고리즘을 이용해 와인의 품질이 좋은지 나쁜지 예측할 수 있는 모델을 만드는 것.

2. Term Project 데이터셋 소개와 속성 설명

1) 데이터셋 - Red Wine Quality

	A	B	C	D	E	F	G	H	I	J	K	L
1	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
2	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
3	7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	5
4	7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	5
5	11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	6
6	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
7	7.4	0.66	0	1.8	0.075	13	40	0.9978	3.51	0.56	9.4	5
8	7.9	0.6	0.06	1.6	0.069	15	59	0.9964	3.3	0.46	9.4	5
9	7.3	0.65	0	1.2	0.065	15	21	0.9946	3.39	0.47	10	7
10	7.8	0.58	0.02	2	0.073	9	18	0.9968	3.36	0.57	9.5	7

그림 1. Wine Quality 데이터셋

- 데이터 셋의 크기 : Instances - 1599, Attributes - 11+output attribute
- 데이터 속성:
  1 - fixed acidity
  2 - volatile acidity
  3 - citric acid
  4 - residual sugar
  5 - chlorides
  6 - free sulfur dioxide
  7 - total sulfur dioxide
  8 - density
  9 - pH
  10 - sulphates
  11 - alcohol
  12 - quality (score between 0 and 10)
- 데이터 셋 링크 : https://archive.ics.uci.edu/ml/datasets/Wine+Quality

3. 전처리

1) 와인의 품질

해당 데이터에는 와인 품질이 0부터 10까지 평가되어있다. 실제 데이터에는 3부터 8까지의 값들이 있다.

이 프로젝트에서는 와인의 품질이 좋은지 나쁜지 확인하는 것이기 때문에 품질 scale을 이진 분류로 바꿔주어야 한다. 따라서 5.5점을 기준으로 5.5보다 낮으면 bad로, 5.5보다 높으면 good로 구분하도록 하였다. 다음 사진은 전처리 과정 코드이다.

```
## 5.5점을 기준으로 좋은 와인과 나쁜 와인을 구분
bins = (2, 5.5, 8)
group_names = ['bad', 'good']
wine['quality'] = pd.cut(wine['quality'], bins = bins,  labels = group_names)

#quality의 good, bad를 인식할 수 없어서 인코딩 실시
label_quality = LabelEncoder()
wine['quality'] = label_quality.fit_transform(wine['quality'])
```

그림 2. 전처리 과정

다음 사진은 전처리 이후 데이터가 어떻게 나누어져 있는지 보여준다. 855개의 좋은 와인이 있고, 744개의 나쁜 와인이 있는 것을 알 수 있다.

```
print(wine['quality'].value_counts())
```

```
1    855
0    744
Name: quality, dtype: int64
```

그림 3. 전처리 이후 결과

2) 척도 표준화

변수 간의 척도가 다른 경우 직접 상호 비교를 할 수 없다. 또한, 척도가 달라서 모수의 왜곡이 생길 수도 있다. 따라서 StandardScaler()를 사용해 변수들의 단위 간 표준화 작업을 시행한다.

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

그림 4. 척도 표준화

4. 데이터셋의 분리

1) 데이터 셋의 분리

데이터 셋 학습을 위해 데이터를 train 데이터와 test 데이터로 분리해준다.

```
X = wine.drop('quality', axis = 1)
y = wine['quality']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
```

그림 5. 데이터 셋 분리

5. 사용한 알고리즘

1) 분류 알고리즘

가. Logistic Regression

로지스틱 회귀분석은 종속변수(Y)와 독립변수(X) 간의 관계를 나타내어 예측모델을 생성한다는 점에서 선형회귀 분석과 비슷하지만, 종속변수(Y)의 결과가 범주형으로 분류 분석에 해당한다. 따라서 와인 품질 분류에 적합한 알고리즘이라고 판단하였다. 다음은 로지스틱 회귀분석 알고리즘을 적용한 코드이다.

Logistic Regression

```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      0.71      0.71       376
           1       0.75      0.75      0.75       424

    accuracy                           0.73       800
   macro avg       0.73      0.73      0.73       800
weighted avg       0.73      0.73      0.73       800
```

```
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.73375
```

그림 6. Logistic Regression 알고리즘 적용 코드

Logistic Regression 모델을 생성하고 fit을 사용해 데이터를 훈련시킨다. Classification_report를 이용하여 머신러닝 분류 모델 평가 지표를 확인하였다. 정확도를 확인해보니 0.733이 나왔다.

나. KNeighbors Classifier

KNeighbors Classifier는 분류 알고리즘 중 하나로, 비슷한 속성(카테고리)을 갖는 데이터끼리 가까이에 위치한다는 것을 이용한 알고리즘이다. k 값에 따라 결과 값이 바뀐다. 이 프로젝트에서는 여러 k 값을 사용해 가장 적합한 k 값을 찾아보았다. 다음은 KNeighbors Classifier 알고리즘을 적용한 코드이다.

KNeighbors Classifier

```
for n_neighbour in [2,3,4,5,6]:
    knn = KNeighborsClassifier(n_neighbour)
    knn.fit(X_train, y_train)
    knn_pred = knn.predict(X_test)
    print("KNeighbor", n_neighbour)
    print("Accuracy:", metrics.accuracy_score(y_test, knn_pred))
    print(classification_report(y_test, knn_pred))
```

그림 7. KNeighbors Classifier 알고리즘 적용 코드

또한, 각 k 값에 따라 달라지는 정확도와 머신러닝 분류 모델 평가 지표를 확인하였다. 그 내용은 다음과 같다.

```
KNeighbor 2
Accuracy: 0.6825
              precision    recall  f1-score   support

           0       0.63      0.79      0.70       376
           1       0.76      0.59      0.66       424

    accuracy                           0.68       800
   macro avg       0.69      0.69      0.68       800
weighted avg       0.70      0.68      0.68       800

KNeighbor 3
Accuracy: 0.6875
              precision    recall  f1-score   support

           0       0.69      0.62      0.65       376
           1       0.69      0.75      0.72       424

    accuracy                           0.69       800
   macro avg       0.69      0.68      0.68       800
weighted avg       0.69      0.69      0.69       800

KNeighbor 4
Accuracy: 0.70375
              precision    recall  f1-score   support

           0       0.67      0.72      0.70       376
           1       0.74      0.69      0.71       424

    accuracy                           0.70       800
   macro avg       0.70      0.70      0.70       800
weighted avg       0.71      0.70      0.70       800

KNeighbor 5
Accuracy: 0.70625
              precision    recall  f1-score   support

           0       0.71      0.64      0.67       376
           1       0.70      0.77      0.73       424

    accuracy                           0.71       800
   macro avg       0.71      0.70      0.70       800
weighted avg       0.71      0.71      0.70       800

KNeighbor 6
Accuracy: 0.705
              precision    recall  f1-score   support

           0       0.68      0.72      0.70       376
           1       0.73      0.70      0.71       424

    accuracy                           0.70       800
   macro avg       0.70      0.71      0.70       800
weighted avg       0.71      0.70      0.71       800
```

그림 8. k 값에 따른 정확도와 모델 평가 지표

내용을 확인해보면 k 값이 5일 때가 0.706으로 가장 정확도가 높고, 다음은 6일 때가 0.705로 높았다. 5일 때와 6일 때의 정확도가 크게 차이 나지 않으므로 다음에 Roc Curve를 그려 한 번 더 확인해보도록 하겠다.

2) 앙상블 알고리즘

가. Random Forest

Random Forest는 기본적으로 Ensemble learning의 "다양한 모델"이라는 기본 컨셉에 충실한 모델이다. Random Forest는 반복 복원추출을 진행하고, 변수를 random 하게 추출하여 다양한 모델을 만드는 것이다. 이 변수를 random 추출하여 모델을 다양하게 만드는 과정을 통해서 base learner 간 공분산을 줄이는 효과를 기대한다. 이 프로젝트에서 사용한 데이터가

약 1600개이므로, 앙상블 알고리즘 중에서 변수를 랜덤하게 추출하는 Random Forest 알고리즘이 적합할 것으로 판단하였다. 다음은 Random Forest 알고리즘을 적용한 코드이다.

Random Forest

```
random = RandomForestClassifier(n_estimators=200)
random.fit(X_train, y_train)

pred_random = random.predict(X_test)

print(classification_report(y_test, pred_random))

print("Accuracy:", metrics.accuracy_score(y_test, pred_random))
```

```
              precision    recall  f1-score   support

           0       0.74      0.74      0.74       376
           1       0.77      0.77      0.77       424

    accuracy                           0.75       800
   macro avg       0.75      0.75      0.75       800
weighted avg       0.75      0.75      0.75       800

Accuracy: 0.75375
```

그림 9. Random Forest 알고리즘 적용 코드

n_estimators의 값을 100, 300, 500으로도 바꾸어 보았지만 모두 정확도가 비슷하게 나와서 200을 선택하였다. 또한, max_depth를 설정하고 여러 값을 주어도 정확도가 크게 달라지지 않아 max_depth는 생략하였다. Random Forest 알고리즘을 공부하다 보니, feature_importances_를 이용해 사용한 변수의 중요도를 확인할 수 있다는 것을 알게 되었다. 따라서 변수의 중요도를 시각화해 보았다. 다음은 변수 중요도 시각화 결과이다.

```
importances = random.feature_importances_
std = np.std([tree.feature_importances_ for tree in random.estimators_], axis = 0)
indices = np.argsort(importances)[::-1]

for f in range(X.shape[1]):
    print("{}. feature {} ({:.3f})".format(f+1, X.columns[indices][f], importances[indices][f]))

plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices])
plt.xticks(range(X.shape[1]), X.columns[indices], rotation=45)
plt.xlim([-1, X.shape[1]])
plt.show()
```

그림 10. 변수 중요도 시각화 코드

1. feature alcohol (0.187)
2. feature sulphates (0.144)
3. feature volatile acidity (0.117)
4. feature total sulfur dioxide (0.092)
5. feature density (0.080)
6. feature chlorides (0.072)
7. feature citric acid (0.068)
8. feature fixed acidity (0.064)
9. feature pH (0.063)
10. feature free sulfur dioxide (0.061)
11. feature residual sugar (0.052)

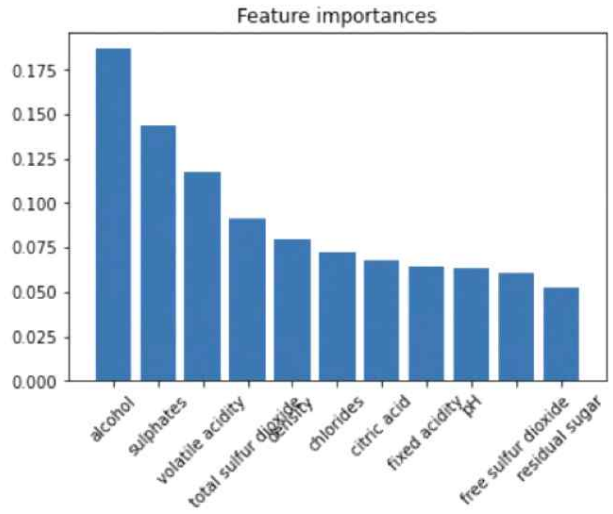


그림 11. 변수 중요도 시각화 결과

시각화한 결과를 확인해보면, Random Forest에서는 알콜 농도와 sulphates(황산염)를 가장 중요한 변수로 판단하여 학습한 것을 알 수 있다.

6. 시각화

1) Roc Curve

이 프로젝트에서는 총 3가지 알고리즘을 적용해 데이터를 학습시켰다. Knn 알고리즘을 k 값에 따라 학습시킨 것까지 포함하면 총 7가지이다. 이 알고리즘 중에서 어떤 알고리즘이 가장 와인 데이터와 잘 맞는 알고리즘인지 시각화하여 확인해보도록 하겠다. 데이터 마이닝 수업을 들으며 했던 HW7에서 Roc Curve를 공부하였다. 그 내용을 이 프로젝트에 적용해보았다.

Roc curve

```
from scipy import interp
from sklearn.metrics import roc_curve, auc
import numpy

def avg_roc(cv, estimator, data, target, pos_label):
    mean_fpr = np.linspace(0, 1, 100) # = [0.0, 0.01, 0.02, 0.03, ... , 0.99, 1.0]
    tprs = []
    aucs = []

    for train_indices, test_indices in cv.split(data, target):
        train_data = data.iloc[train_indices]
        train_target = target[train_indices]
        estimator.fit(train_data, train_target)

        test_data = data.iloc[test_indices]
        test_target = target[test_indices]
        decision_for_each_class = estimator.predict_proba(test_data)#have to use predict_proba or decision_function

        fpr, tpr, thresholds = roc_curve(test_target, decision_for_each_class[:,1], pos_label=pos_label)
        tprs.append(interp(mean_fpr, fpr, tpr))
        tprs[-1][0] = 0.0 # tprs[-1] access the last element
        aucs.append(auc(fpr, tpr))

        #plt.plot(fpr, tpr)# plot for each fold

    mean_tpr = np.mean(tprs, axis=0)
    mean_tpr[-1] = 1.0 # set the last tpr to 1
    mean_auc = auc(mean_fpr, mean_tpr)
    std_auc = np.std(aucs)

    return mean_fpr, mean_tpr, mean_auc, std_auc
```

그림 12. Roc Curve 생성 함수 코드

위의 코드에서 쓰인 avg_roc 함수는 HW7에서 사용했던 함수를 그대로 가져와 적용했다. 다음은 이 함수를 프로젝트에서 쓰인 여러 알고리즘에 적용해 Roc Curve를 시각화한 내용이다.

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

preprocessor = ColumnTransformer(
    transformers=[('scaler', StandardScaler(), ['fixed acidity','volatile acidity','citric acid','residual sugar','chlorides',
                                                'free sulfur dioxide','total sulfur dioxide','density','pH','sulphates','alcohol']) ])

pipeline = Pipeline([ ('preprocessing', preprocessor), ('estimator', None) ])

plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random', alpha=.8) # draw diagonal

# KNN
for n_neighbour in [2,3,4,5,6]:
    mean_fpr, mean_tpr, mean_auc, std_auc = avg_roc(cv, KNeighborsClassifier(n_neighbour), X, y, 1)
    plt.plot(mean_fpr, mean_tpr, label='{}-NN (AUC: {:.3f} $\pm$ {:.3f})'.format(n_neighbour, mean_auc, std_auc))

#LogisticRegression
mean_fpr, mean_tpr, mean_auc, std_auc = avg_roc(cv, LogisticRegression(), X, y, 1)
plt.plot(mean_fpr, mean_tpr, label='LogisticRegression (AUC: {:.3f} $\pm$ {:.3f})'.format(mean_auc, std_auc))

#RandomForest
mean_fpr, mean_tpr, mean_auc, std_auc = avg_roc(cv, random, X, y, 1)
plt.plot(mean_fpr, mean_tpr, label='RandomForest (AUC: {:.3f} $\pm$ {:.3f})'.format(mean_auc, std_auc))

plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.legend()
plt.show()
```

그림 13. Roc Curve 시각화 코드

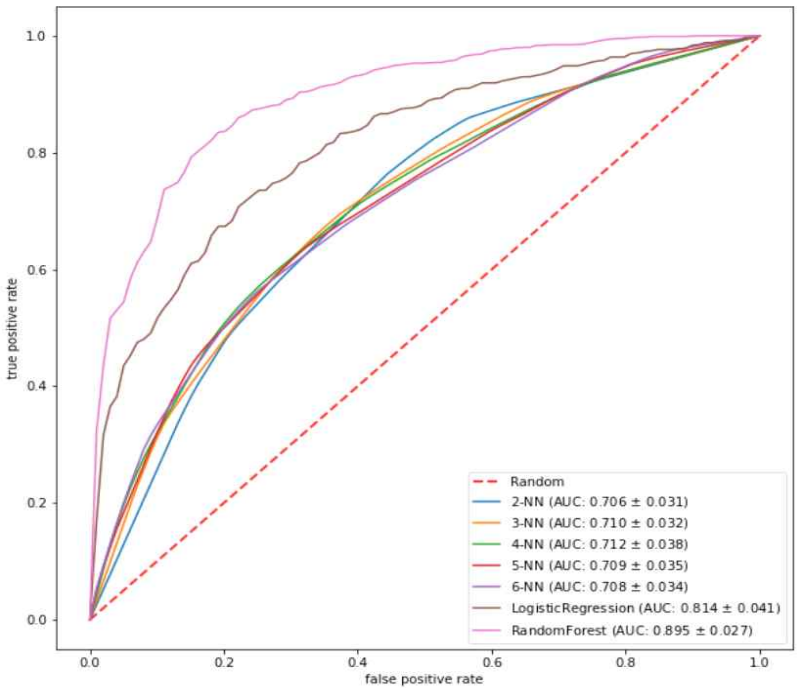


그림 14. Roc curve 시각화 결과

Roc curve를 확인해보면, 이 데이터에 가장 적합한 알고리즘은 Random Forest 알고리즘과 Logistic Regression 알고리즘인 것을 확인할 수 있다. 하지만 여전히 KNeighbors Classifier 알고리즘 중에서는 어떤 k 값이 가장 적절한지 확인하기 힘들다. 따라서 Confusion Matrix를 사용해 알아보도록 하겠다.

2) Confusion Matrix

다음은 Confusion Matrix를 시각화하는 함수와 이를 와인 데이터에 적용한 코드이다.

```
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

그림 15. Confusion Matrix 시각화 함수

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

plt.figure(figsize=(10,5))

classes = y.unique()

# KNN
plt.subplot(1,4,1)
pipeline.set_params(estimator=KNeighborsClassifier(6))
prediction = cross_val_predict(pipeline, X, y, cv=cv)
cnf_matrix = confusion_matrix(y, prediction, labels=classes)
plot_confusion_matrix(cnf_matrix, classes=classes, title='6-NN Classifier')

plt.subplot(1,4,2)
pipeline.set_params(estimator=KNeighborsClassifier(5))
prediction = cross_val_predict(pipeline, X, y, cv=cv)
cnf_matrix = confusion_matrix(y, prediction, labels=classes)
plot_confusion_matrix(cnf_matrix, classes=classes, title='5-NN Classifier')

#LogisticRegression
plt.subplot(1,4,3)
pipeline.set_params(estimator=LogisticRegression())
prediction = cross_val_predict(pipeline, X, y, cv=cv)
cnf_matrix = confusion_matrix(y, prediction, labels=classes)
plot_confusion_matrix(cnf_matrix, classes=classes, title='LogisticRegression')

# RandomForest
plt.subplot(1,4,4)
pipeline.set_params(estimator=random)
prediction = cross_val_predict(pipeline, X, y, cv=cv)
cnf_matrix = confusion_matrix(y, prediction, labels=classes)
plot_confusion_matrix(cnf_matrix, classes=classes, title='RandomForest')
```

그림 16. Confusion Matrix 시각화 적용 코드

위에서 사용한 plot_confusion_matrix 함수 또한 HW7에서 사용했던 함수를 그대로 가져와 적용하였다. 다음은 Confusion Matrix를 시각화한 결과이다.

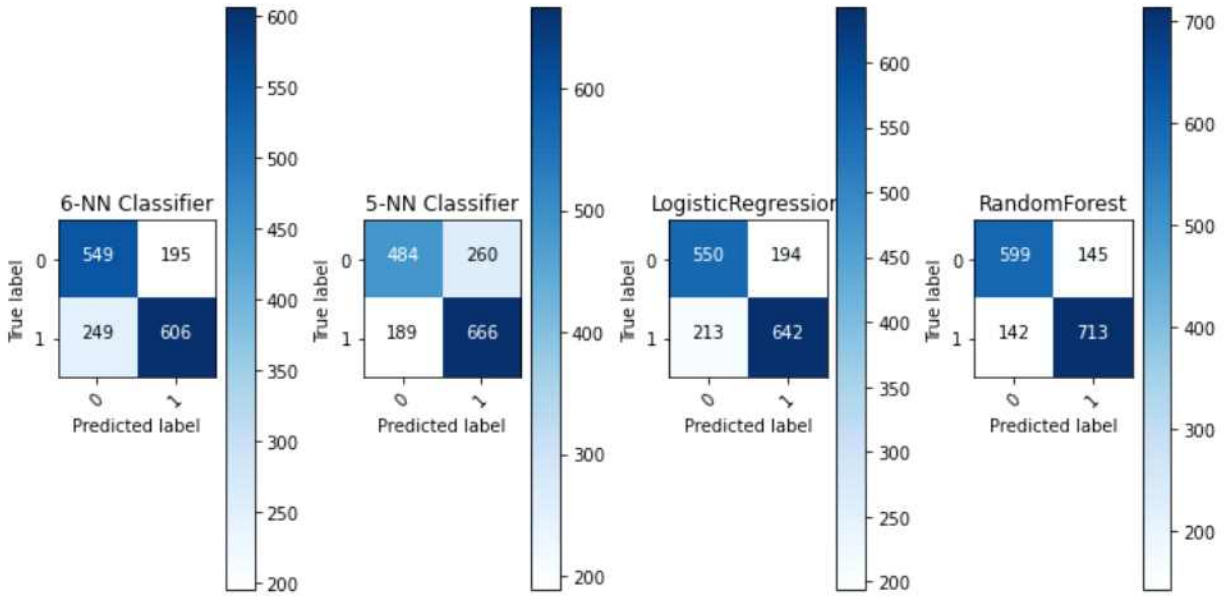


그림 17. Confusion Matrix 시각화 결과

Confusion Matrix를 확인해보면 Random Forest에서 예측한 정답은 1312개, 오답은 287개로 가장 적합한 알고리즘이라는 것을 확인할 수 있다. 또한, Logistic Regression은 1192개의 정답을, 413개의 오답을 예측하여서 두 번째로 적합한 알고리즘이라는 것을 확인할 수 있다. 그리고 k 값이 6일 때의 정답이 1155개, 오답이 444개로 k 값이 5일 때의 정답인 1150개, 오답 449개보다 근소하게 더 적합한 알고리즘이라는 것을 확인할 수 있었다.

7. 결과 해석

이 프로젝트는 와인의 품질이 좋은지 나쁜지 예측하는 모델을 만드는 프로젝트이다. 레드와인의 품질 데이터를 가지고 진행하였고, Logistic Regression, Random Forest, KNeighbor Classifier 총 3개의 알고리즘을 적용하였다.

Logistic Regression 알고리즘은 정확도는 0.734로 나머지 알고리즘 중에서 가장 낮았지만, Roc Curve와 Confusion Matrix를 확인하였을 때, 이 데이터에 가장 적합한 알고리즘이라고 판단하였다.

Random Forest 알고리즘의 정확도는 0.754였고, Roc Curve와 Confusion Matrix에서 모두 두 번째로 적합한 알고리즘이라고 판단되었다.

KNeighbor Classifier 알고리즘은 k 값에 따라 변하는 알고리즘이기에 여러 k 값을 적용해 진행하였다. 정확도에서는 5일 때가 0.706으로 6일 때 0.705보다 근소하게 높았지만, Roc Curve와 Confusion Matrix를 통해 확인한 결과, k 값이 6일 때가 더 적합한 알고리즘으로 판단되었다.

8. HW7 - 4.2

1) 데이터 셋

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_commitment
0	<0	6.0	critical/other existing credit	radio/tv	1169.0	no known savings	>=7	4.0
1	0<=X<200	48.0	existing paid	radio/tv	5951.0	<100	1<=X<4	2.0
2	no checking	12.0	critical/other existing credit	education	2096.0	<100	4<=X<7	2.0
3	<0	42.0	existing paid	furniture/equipment	7882.0	<100	4<=X<7	2.0
4	<0	24.0	delayed previously	new car	4870.0	<100	1<=X<4	3.0

그림 18. credit 데이터 셋

사람들의 신용 정보에 관한 데이터이다. 신용 경력, 지속기간, 신용의 양, 재산, 직업, 월급, 나이 등에 관한 데이터가 포함되어 있다.

이 데이터는 신용 정보가 좋고 나쁨으로 구분되어있다. 그 개수를 확인하도록 하겠다.

```
class_dist = pd.Series(credit_target).value_counts()
plt.bar(class_dist.index, class_dist)
plt.ylabel("Frequency")
plt.show()
```

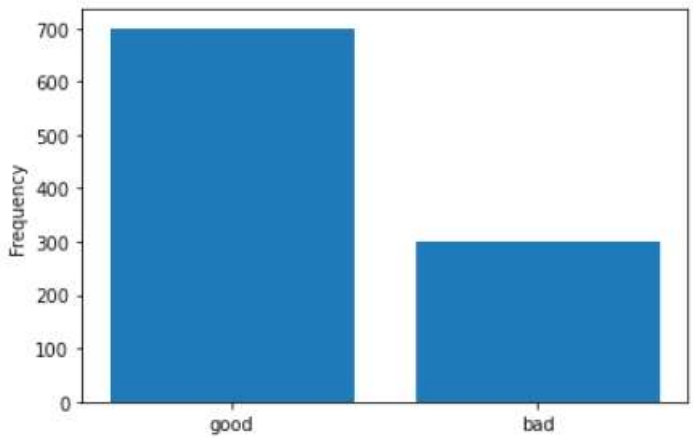


그림 19. credit 데이터 셋의 좋고 나쁨의 분배

2) 데이터 전처리

credit 데이터는 많은 부분이 문자열로 되어있다. 이렇게 되면 알고리즘에 적용해 훈련할 수 없으므로 숫자 데이터로 변형시켜야 한다. Standard Scaler() 와 OneHotEncoder()를 사용해 전처리하였다. 다음은 전처리 과정과 결과이다.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
from sklearn.compose import ColumnTransformer

# think about which features you want to re-scale, encode using one-hot encoding or encode using ordinal encoding
# then, create a ColumnTransformer to execute this preprocessing for you

preprocessor = ColumnTransformer(
    transformers=[
        # use StandardScaler for Temperature and Humidity
        ('scaler', StandardScaler(), ['duration', 'credit_amount','installment_commitment','residence_since','age','exi
        # use OneHotEncoder for Outlook and Wind
        ('encoder', OneHotEncoder(), ["checking_status", "credit_history", "purpose", "savings_status", "employment", "

preprocessed = pd.DataFrame(preprocessor.fit_transform(credit_data))
display(preprocessed.head())

pipeline = Pipeline([("preprocessor", preprocessor), ("classifier", StandardScaler())])
```

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54	55	56	57	58	59	60
0	-1.236478	-0.745131	0.918477	1.046987	2.766456	1.027079	-0.428290	0.0	1.0	0.0	...	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0
1	2.248194	0.949817	-0.870183	-0.765977	-1.191404	-0.704926	-0.428290	1.0	0.0	0.0	...	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
2	-0.738668	-0.416562	-0.870183	0.140505	1.183312	-0.704926	2.334869	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0
3	1.750384	1.634247	-0.870183	1.046987	0.831502	-0.704926	2.334869	0.0	1.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
4	0.256953	0.566664	0.024147	1.046987	1.535122	1.027079	2.334869	0.0	1.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0

그림 20. 전처리 결과

3) 알고리즘 적용 및 시각화

전처리한 데이터를 가지고 KNeighborsClassifier, DecisionTreeClassifier, GaussianNB에 훈련시켜 보았다. 이 과정에서 10-fold cross-validation을 함께 사용하였다. 또한, 사용한 알고리즘들을 비교 분석하기 위해 Roc Curve를 시각화하였다. 다음은 알고리즘을 적용하고 Roc Curve를 시각화한 결과이다. Roc Curve를 시각화하면서 사용한 avg_roc 함수는 위 6.1) 부분에서 설명했으므로 생략하도록 하겠다.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import GaussianNB

# define the cross validation folds
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# create the pipeline, we will set the estimator later
pipeline = Pipeline([ ('preprocessing', preprocessor), ('estimator', None) ])

# setup a figure
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random', alpha=.8) # draw diagonal

# KNN
#TODO: INSERT YOUR CODE HERE!
for n_neighbour in [2,3,4,5,6]:
    #TODO: INSERT YOUR CODE HERE!
    mean_fpr, mean_tpr, mean_auc, std_auc = avg_roc(cv, KNeighborsClassifier(n_neighbour), preprocessed, credit_target, 'good')
    plt.plot(mean_fpr, mean_tpr, label='{}-NN (AUC: {:.3f} $\pm$ {:.3f})'.format(n_neighbour, mean_auc, std_auc))

# Decision Tree
#TODO: INSERT YOUR CODE HERE!
mean_fpr, mean_tpr, mean_auc, std_auc = avg_roc(cv, DecisionTreeClassifier(), preprocessed, credit_target, 'good')
plt.plot(mean_fpr, mean_tpr, label='DecisonTree (AUC: {:.3f} $\pm$ {:.3f})'.format(mean_auc, std_auc))

#Naive Bayes
mean_fpr, mean_tpr, mean_auc, std_auc = avg_roc(cv, GaussianNB(), preprocessed, credit_target, 'good')
plt.plot(mean_fpr, mean_tpr, label='GaussianNB (AUC: {:.3f} $\pm$ {:.3f})'.format(mean_auc, std_auc))

# # show the plot
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.legend()
plt.show()
```

그림 21. 알고리즘 적용 후 Roc Curve 시각화 코드

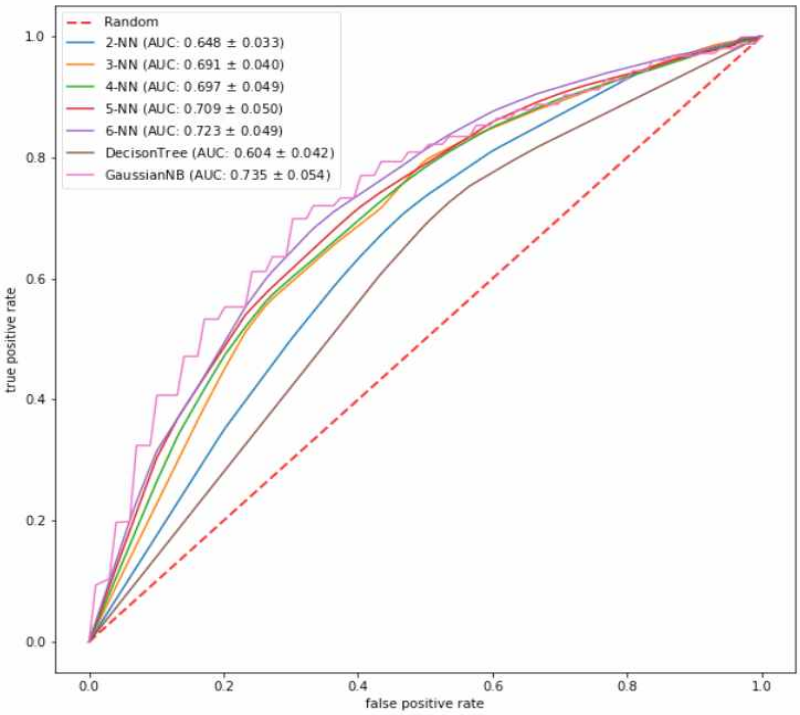
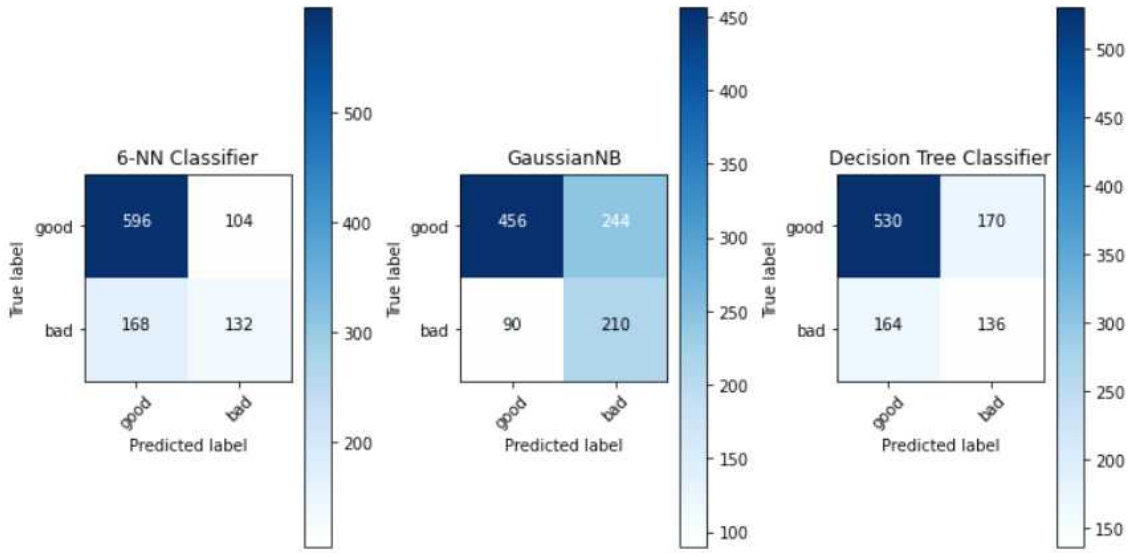


그림 22. Roc Curve 시각화 결과

Roc Curve를 보면 GaussianNB가 가장 적합한 알고리즘이라고 판단할 수 있고, KNeighbors Classifier에서 k 값이 6일 때가 두 번째로 적합한 알고리즘이다. Decision Tree Classifier는 매우 적합하지 않은 알고리즘으로 보인다. Confusion Matrix와 classification_report를 이용해 한 번 더 확인해보도록 하겠다.



Classification Report for 6-NN Classifier:

	precision	recall	f1-score	support
bad	0.56	0.44	0.49	300
good	0.78	0.85	0.81	700
accuracy			0.73	1000
macro avg	0.67	0.65	0.65	1000
weighted avg	0.71	0.73	0.72	1000

Classification Report for GaussianNB:

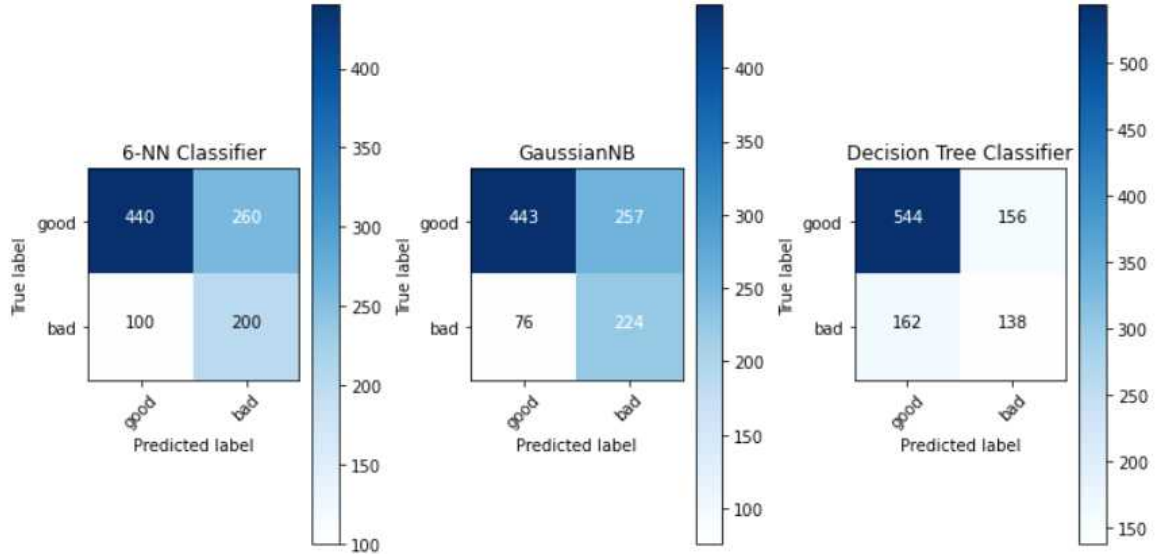
	precision	recall	f1-score	support
bad	0.46	0.70	0.56	300
good	0.84	0.65	0.73	700
accuracy			0.67	1000
macro avg	0.65	0.68	0.64	1000
weighted avg	0.72	0.67	0.68	1000

Classification Report for Decision Tree Classifier:

	precision	recall	f1-score	support
bad	0.44	0.45	0.45	300
good	0.76	0.76	0.76	700
accuracy			0.67	1000
macro avg	0.60	0.61	0.60	1000
weighted avg	0.67	0.67	0.67	1000

그림 23. Roc Curve 시각화 결과

이 과정에서 bad를 오버 샘플링 해보도록 하겠다. 다음은 오버 샘플링 결과이다.



Classification Report for 6-NN Classifier:

	precision	recall	f1-score	support
bad	0.43	0.67	0.53	300
good	0.81	0.63	0.71	700
accuracy			0.64	1000
macro avg	0.62	0.65	0.62	1000
weighted avg	0.70	0.64	0.65	1000

Classification Report for GaussianNB:

	precision	recall	f1-score	support
bad	0.47	0.75	0.57	300
good	0.85	0.63	0.73	700
accuracy			0.67	1000
macro avg	0.66	0.69	0.65	1000
weighted avg	0.74	0.67	0.68	1000

Classification Report for Decision Tree Classifier:

	precision	recall	f1-score	support
bad	0.47	0.46	0.46	300
good	0.77	0.78	0.77	700
accuracy			0.68	1000
macro avg	0.62	0.62	0.62	1000
weighted avg	0.68	0.68	0.68	1000

그림 24. 오버 샘플링한 결과

bad를 오버 샘플링 하였지만, f1-score 값이 조금 높아진 것 말고는 큰 변화가 없었다.

사용 사례별 평가를 모형화하려면 모든 누락 분류 비용을 계산 해야 한다. 좋은 고객에게 크레딧을 거절하면 1유닛이 손실되고, 나쁜 고객에게 크레딧을 주면 100유닛이 손실된다고 가정하여 비용 매트릭스를 설정하였다. 다음은 그 결과이다.

```
from sklearn.metrics import confusion_matrix

# create the predictions here
# prediction = ...
pipeline.set_params(estimator=KNeighborsClassifier(6))
prediction = cross_val_predict(pipeline, credit_data, credit_target, cv=cv)

# You can use the following code to calculate the cost
cm = confusion_matrix(credit_target, prediction, labels=credit_target.unique())
cost = cm[0][1] * 100 + cm[1][0] * 1
acc = accuracy_score(credit_target, prediction)
print("6-NN with accuracy of {} and cost {}".format(acc, cost))
```

```
6-NN with accuracy of 0.639 and cost 26497
```

그림 25. 사용 사례별 평가를 모형화한 결과

9. HW7 - 4.3

1) 데이터 셋

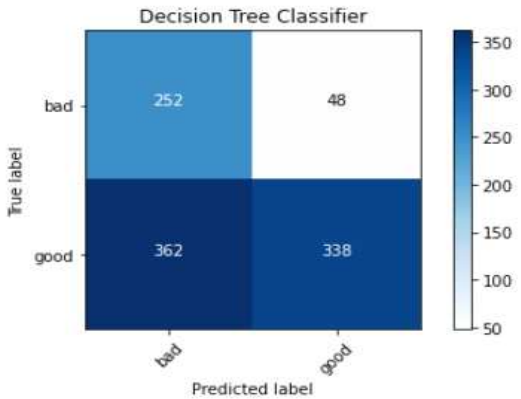
HW7 - 4.3의 데이터 셋은 4.2의 데이터 셋과 같은 credit 데이터 셋이다.

2) 알고리즘 적용

4.2를 확장하여 GridSearchCV를 이용하여 Decision Tree Classifier에 더 적합한 구성을 찾아보려고 한다. 다음은 이에 해당하는 코드와 결과이다.

```
from sklearn.model_selection import GridSearchCV

# define the parameter grid
#TODO: INSERT YOUR CODE HERE
parameters = {
    'estimator__criterion':['gini', 'entropy'],
    'estimator__max_depth':[ 2, 3, 4, 5, None],
    'estimator__min_samples_split' :[2,3,4,5]
}

# define the folds for the cross validation
stratified_cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# create a scorer for the grid search
cost_score = make_scorer(cost_function, greater_is_better=False)

# create the grid search estimator
#TODO: INSERT YOUR CODE HERE
#grid_search_estimator = ...
grid_search_estimator = GridSearchCV(pipeline, parameters, scoring=cost_score, cv=stratified_cv)

# cross-validate
#TODO: INSERT YOUR CODE HERE
#prediction = ...
prediction = cross_val_predict(grid_search_estimator, credit_data, credit_target, cv=stratified_cv)

# calculate costs
cm = confusion_matrix(credit_target, prediction, labels=label_order)
cost = cost_function(credit_target, prediction)
acc = accuracy_score(credit_target, prediction)

print("Optimised Decision Tree with accuracy of {} and cost {}".format(acc, cost))
plot_confusion_matrix(cm, classes=label_names, title='Decision Tree Classifier')
plt.show()
print(classification_report(credit_target, prediction, target_names=label_names))

# fit the grid search (= determine the optimal parameters)
#TODO: INSERT YOUR CODE HERE
grid_search_estimator.fit(credit_data, credit_target)
print("Optimised Parameters: {}".format(grid_search_estimator.best_params_))
```

```
Optimised Decision Tree with accuracy of 0.59 and cost 5162
```



	precision	recall	f1-score	support
bad	0.41	0.84	0.55	300
good	0.88	0.48	0.62	700
accuracy			0.59	1000
macro avg	0.64	0.66	0.59	1000
weighted avg	0.74	0.59	0.60	1000

```
Optimised Parameters: {'estimator__criterion': 'entropy', 'estimator__max_depth': 2, 'estimator__min_samples_split': 3}
```

그림 26. GridSearchCV 이용한 코드와 결과

GridSearchCV는 분류 알고리즘이나 회귀 알고리즘에 사용되는 하이퍼 파라미터를 차례로 입력해 학습하고 측정을 하면서 가장 좋은 파라미터를 알려준다. grid 파라미터 안에서 집합을 만들고 적용하여 최적화된 파라미터를 뽑아낼 수 있다. GridSearchCV를 사용한 결과를 확인하면, criterion은 entropy를 사용하였고 max_depth는 2를 사용하였고 min_samples_split는 3을 사용한 것을 알 수 있다.

과제에서는 n_jobs=-1을 사용하는 것을 권장하였지만, n_jobs=-1을 사용하면 오류가 나서 사용할 수 없었다. 이를 사용하지 않음으로써 GridSearchCV의 속도가 매우 느려졌지만, 이것의 대체 방안은 찾지 못하였다.

10. 느낀 점

한 학기 동안 공부한 여러 알고리즘을 내가 선택한 데이터에 맞추어서 프로젝트를 진행할 수 있어서 뿌듯했다. 데이터의 특성마다 전처리도 전부 달라지고 알고리즘들을 적용하는 것도 달라서 오류가 자주 났었다. 그 오류들을 해결하는 과정에서 더 많이 배울 수 있었던 것 같다. 때문에 머신러닝 분야에도 많은 흥미를 느꼈다. HW7을 진행하면서 아직 익숙하지 않은 부분들이 많아서 내가 배울 것이 너무나도 많은 것을 느꼈다. 이번 학기를 잘 마무리한 후에 겨울방학 때 머신러닝에 대해 더 심도 있는 공부를 진행해야겠다고 마음먹었다.

11. 참고한 정보

- https://scikit-learn.org/stable/index.html - 여러 알고리즘의 사용 방법이나, 설정 방법, 파라미터들을 많이 참고하였다.
- https://stackoverflow.com/ - 오류가 발생했을 때 가장 많이 참고하였고 도움이 되었다.
- 그 외에도 머신러닝과 데이터 마이닝을 공부하는 많은 사람이 남긴 블로그나 기록들을 많이 참고하였다.