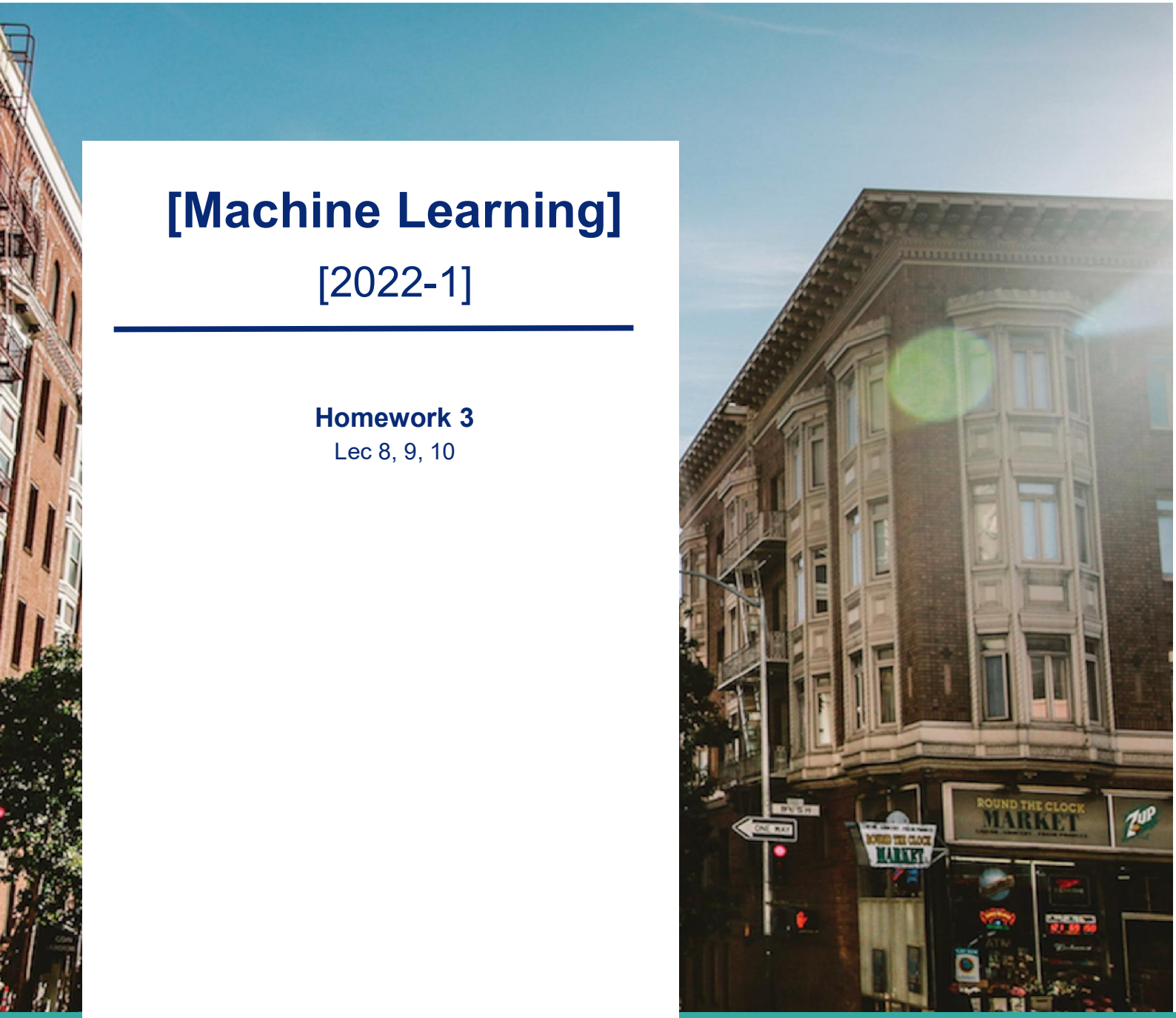


[Machine Learning]

[2022-1]

Homework 3

Lec 8, 9, 10

[DATE] 2022.05.19

Student ID : 2017112138

Name : 정여준

Professor : Juntae Kim

1. Explain what ‘Feature Selection’ and ‘Dimensionality Reduction’ are, and why they are needed in machine learning. You can use examples for explanation. (10pts)

Your Answer
Feature Selection: 모델링 시 raw data 의 모든 feature 를 사용하는 것은 비효율적이기 때문에 일부 필요한 feature 만 선택해서 사용하는 것이다. Feature Selection 을 하면 사용자가 해석하기 쉽게 모델을 단순화할 수 있고 훈련 시간의 축소, 차원의 저주 방지, 일반화와 같은 장점이 있기 때문에 머신러닝에서 모델성능을 높이기 위해 필수적인 기술이다. Dimensionality Reduction: 차원 축소는 데이터의 의미를 제대로 표현하는 특징을 추려내는 것이다. 즉 더 좋은 특징만 가지고 사용하는 것이다. 차원이 증가하면 그것을 표한하기 위한 데이터 양이 기하급수적으로 증가한다. 그렇기 때문에 너무 고차원의 데이터들은 의미를 제대로 표현하기 어렵다. 그래서 차원 축소를 사용하는 것이다.

2. Explain what ‘Bias’ and ‘Variances’ are and how they affect the performance of machine learning models. (10pts)

Your Answer
bias 는 모델을 통해 얻은 예측 값과 실제 정답과의 차이의 평균을 나타낸다. 즉, 예측 값이 실제 정답 값과 얼만큼 떨어져 있는 지 나타낸다. Variance 는 예측한 값들의 다양성으로 다양한 데이터 셋에 대하여 예측 값이 얼만큼 변화할 수 있는 지에 대한 양의 개념이다. 이는 모델이 얼만큼 flexibility 를 가지는 지에 대한 의미로도 사용된다. Bias 가 머신러닝 모델의 성능에 미치는 영향: High bias -> underfitting -> increase features, decrease regularization Variance 가 머신러닝 모델의 성능에 미치는 영향: High variance -> overfitting -> Decrease features, increase regularization

3. Explain what ‘ROC curve’ is, and when they are needed in performance analysis of machine learning model. (10pts)

Your Answer
ROC curve 란 모델의 판단 기준을 연속적으로 바꾸면서 측정했을 때 FPR 과 TPR 의 변화를 나타낸 것이다. 여기서 TPR 은 1 인 케이스에 대해 1 로 올바르게 예측하는 비율(Sensitivity)이고 FPR 은 0 인 케이스에 대해 1 로 틀리게 예측하는 비율(1-Specificity)이다. ROC curve 는 머신러닝 모델의 유용성을 판단하거나 정확도를 평가하는데 사용되고 기준점을 설정하는 경우에도 활용될 수 있다.

4. Answer following questions. (20pts)

4-1. Perform one-hot encoding on the class of Iris data.

Expected Output

	0	1	2	3	4
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```
print(y[0])
print(y[50])
print(y[100])
```

```
[1. 0. 0.]
[0. 1. 0.]
[0. 0. 1.]
```

Code

```
from sklearn.preprocessing import OneHotEncoder
from sklearn import datasets

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

ohe=OneHotEncoder(sparse=False)
reshaped=y.reshape(len(y), 1)
```

```
y=ohe.fit_transform(reshaped)

print(y[0])
print(y[50])
print(y[100])
```

Result(Captured images)

```
from sklearn.preprocessing import OneHotEncoder
from sklearn import datasets

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

ohe=OneHotEncoder(sparse=False)
reshaped=y.reshape(len(y), 1)
y=ohe.fit_transform(reshaped)

print(y[0])
print(y[50])
print(y[100])
```

```
[1. 0. 0.]
[0. 1. 0.]
[0. 0. 1.]
```

Description

one-hot encoding 을 하기 위해서 OneHotEncoder()를 사용하고 이를 사용하여 fit 하고 transform 하기 위해서 y 를 reshape 해준다.

4-2. Perform Standardization on the Iris data(except class. split train and test set)

Expected Output

	0	1	2	3	4
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```
X_train_std
```

```
array([[-0.923122  ,  1.61656532, -1.05236498, -1.06144375],
       [-0.923122  ,  0.96288358, -1.33253314, -1.19475906],
       [-0.21128986,  2.9239288 , -1.27649951, -1.06144375],
       [ 2.16148395, -0.56237382,  1.63724937,  1.07160111],
```

```
X_test_std
```

```
array([[-0.44856724,  0.96288358, -1.38856677, -1.32807436],
       [-1.75359284,  0.30920184, -1.38856677, -1.32807436],
       [ 1.21237443,  0.09130793,  0.90881215,  1.20491641],
       [ 0.97509705,  0.52709575,  1.07691305,  1.73817763],
```

Code

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0, stratify=y)
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

Result(Captured images)

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
X_train_std
```

Python

```
Output exceeds the size limit. Open the full output data in a text editor
array([[-0.923122  ,  1.59577792, -1.0511    , -1.06144375],
       [-0.923122  ,  0.94633342, -1.33107336, -1.19475906],
       [-0.21128986,  2.89466693, -1.27507869, -1.06144375],
       [ 2.16148395, -0.56903709,  1.63664429,  1.07160111],
       [ 0.61918098, -0.56903709,  1.02070289,  1.20491641],
```

```
X_test_std
```

Python

```
Output exceeds the size limit. Open the full output data in a text editor
array([[-0.44856724,  0.94633342, -1.38706803, -1.32807436],
       [-1.75359284,  0.29688892, -1.38706803, -1.32807436],
       [ 1.21237443,  0.08040741,  0.90871354,  1.20491641],
       [ 0.97509705,  0.51337042,  1.07669756,  1.73817763],
       [ 1.33101312,  0.29688892,  0.51675084,  0.27170929],
       [ 0.50054228, -0.78551859,  0.62874018,  0.8049705 ],
       [-0.44856724, -1.00200009,  0.34876682,  0.00507868],
```

Description

train_test_split 을 사용하여 train 과 test 를 나누는데 test size 는 30%로 설정하고 진행한다. 표준화하기 위해 StandardScaler 를 사용한다. 스케일러를 사용하여 X_train 을 fit 하고 transform 함수를 통해 X_train 을 표준화한 X_trains_std 와 X_test 를 표준화한 X_test_std 를 생성한다.

4-3. Perform PCA on the iris data, check the explained variance ratio, and choose 2 components. Plot the data with 1) petal length and petal width, and 2) PC1 and PC2

Expected Output

```
pca.explained_variance_ratio_
```

```
array([0.73388509, 0.22547013, 0.03516739, 0.00547739])
```

```
X_train_pca
```

```
array([[-2.14486538e+00,  1.00530441e+00],
       [-2.19536586e+00,  3.91637317e-01],
       [-2.28031379e+00,  2.47772726e+00],
       [ 2.82749300e+00,  4.78884590e-01],
```

```
X_test_pca
```

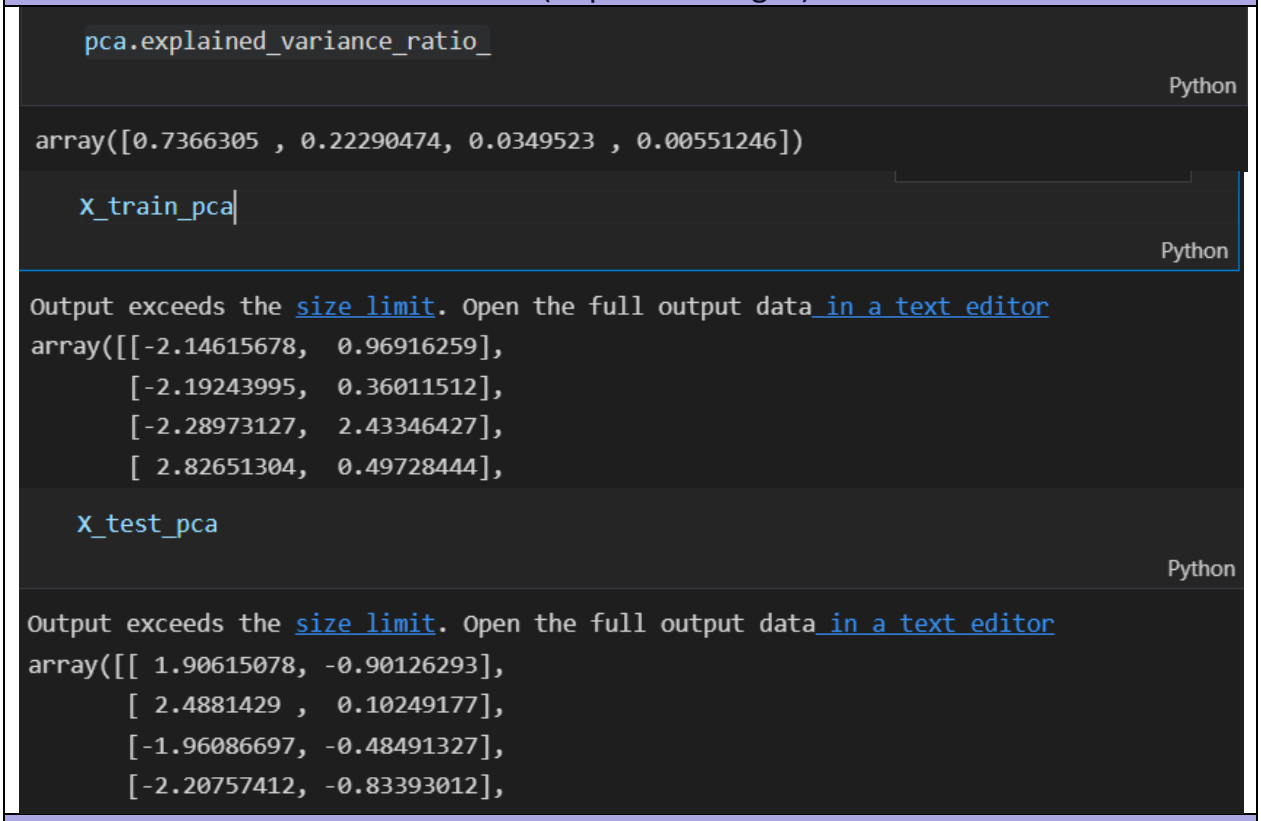
```
array([[ 1.90725282, -0.90395608],
       [ 2.48651617,  0.10200288],
       [-1.96227467, -0.48731458],
       [-2.20910465, -0.84096144],
```

Code

```
from sklearn.decomposition import PCA
pca = PCA()
X_train_pca = pca.fit_transform(X_train_std)
```

```
pca.explained_variance_ratio_
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.fit_transform(X_test_std)
```

Result(Captured images)



Description

PCA 를 통해 iris data 의 주성분 분석을 실시하고 explained_variance_ratio_를 통해 분산도를 확인한다. 2 개의 주성분으로 분석한 데이터를 만들기 위해서 PCA(n_components=2)를 사용한다. 이 상태에서 fit_transform 하여 X_train_pca 와 X_test_pca 를 구한다.

5. Apply Bagging and Boosting algorithms on Iris dataset. Set the base_estimator to “DecisionTreeClassfier”. We recommend you use “BaggingClassifier” and “AdaBoostClassifier” in Scikit-learn. (20pts)

Iris data Setting(Use this Code)

```
import numpy as np
import pandas as pd
df = pd.read_csv('https://archive.ics.uci.edu/ml/'
      'machine-learning-databases/iris/iris.data', header=None)
X = df.iloc[50:150, [2, 3]].values
y = df.iloc[50:150, 4].values
y = np.where(y == 'Iris-virginica', -1, 1)
```

Expected Output

```
print("Decision Tree train/test accuracies %.3f/%.3f"
    % (tree_clf.score(X_train, y_train), tree_clf.score(X_test, y_test)))
print("Bagging train/test accuracies %.3f/%.3f"
    % (bag_clf.score(X_train, y_train), bag_clf.score(X_test, y_test)))
print("Adaboost train/test accuracies %.3f/%.3f"
    % (boost_clf.score(X_train, y_train), boost_clf.score(X_test, y_test)))
```

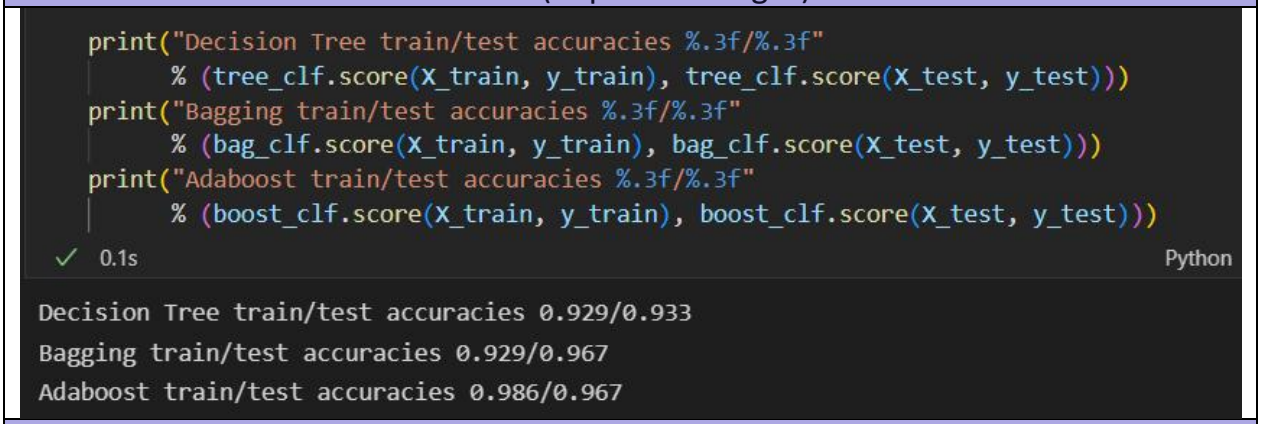
```
Decision Tree train/test accuracies 0.929/0.933
Bagging train/test accuracies 0.929/0.967
Adaboost train/test accuracies 0.986/0.967
```

Code

```
import numpy as np
import pandas as pd
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data', header=None)
X = df.iloc[50:150, [2, 3]].values
y = df.iloc[50:150, 4].values
y = np.where(y == 'Iris-virginica', -1, 1)

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
y = le.fit_transform(y)
X_train, X_test, y_train, y_test
=train_test_split(X,y,test_size=0.3,random_state=1,stratify=y)

from sklearn.tree import DecisionTreeClassifier

tree_clf = DecisionTreeClassifier(max_depth=1)
tree_clf.fit(X_train, y_train)

from sklearn.ensemble import BaggingClassifier

bag_clf = BaggingClassifier(base_estimator=tree_clf)
bag_clf.fit(X_train, y_train)

from sklearn.ensemble import AdaBoostClassifier

boost_clf = AdaBoostClassifier(base_estimator=tree_clf)
boost_clf.fit(X_train, y_train)
print("Decision Tree train/test accuracies %.3f/%.3f"
      % (tree_clf.score(X_train, y_train), tree_clf.score(X_test, y_test)))
print("Bagging train/test accuracies %.3f/%.3f"
      % (bag_clf.score(X_train, y_train), bag_clf.score(X_test, y_test)))
print("Adaboost train/test accuracies %.3f/%.3f"
      % (boost_clf.score(X_train, y_train), boost_clf.score(X_test, y_test)))
```

Result(Captured images)



Description

base_estimator 을 DecisionTreeClassifier 로 설정하기 위해서 max_depth 가 1 인 DecisionTreeClassifier 를 생성하고 이를 base_estimator 로 설정하는 BaggingClassifier 와 AdaBoostClassifier 를 생성하고 train 과 test 의 정확도를 구한다.

6. Apply Logistic Regression and Decision Tree Classifier on 20newsgroups Dataset(Document Classification). (30pts)

20newsgroup dataset is a news dataset consisting of 20 categories. In this question, we only use samples from 4 categories.

Follow this process:

1) Load the data.

```
from sklearn.datasets import fetch_20newsgroups

categories = ['alt.atheism', 'soc.religion.christian',
              'comp.graphics', 'sci.med']

news = fetch_20newsgroups(categories=categories,
                          shuffle=False,
                          random_state=42)
X = news.data
y = news.target
```

```
X[0]
```

```
"From: keith@cco.caltech.edu (Keith Allan Schneider)\nSubject: Re: <
d.caltech.edu\n\nbobbe@vice.ICO.TEK.COM (Robert Beauchaine) writes:\
ral hypothesis, you have to successfully argue that\n>domestication
animals exhibit behaviors not found in the wild.  I\ndon't think tha
years\nto produce certain behaviors, etc.\n\nkeith\n"
```

```
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,
                                                    stratify=y,
                                                    random_state=1)
```

2) Make preprocessor function & porter stemmer tokenizer function.

3) Apply the TF-IDF(TFidfVectorizer) on the data.

- Use the preprocessor function & porter stemmer tokenizer
- Use stop-words
- Drop terms occurred in more than 10% of docs
- Drop terms occurred in less than 10 docs

4) Train machine learning models(Logistic Regression, Decision Tree) using TF-IDF vectors.

5) Predict the categories of following 4 sentences

'The outbreak was declared a global pandemic by the World Health Organization (WHO) on 11 March.',

'Today, computer graphics is a core technology in digitalphotography, film, video games, cell phone and computer displays,and many specialized applications.',

'Arguments for atheism range from philosophical to social and historical approaches.',

'The Bible is a compilation of many shorter books written at different times by a variety of authors, and later assembled into the biblical canon.'

Expected Output

Training a Logistic Regression Model for Document Classification

```
# train using Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(penalty="l2", verbose=1)
lr.fit(train_vector, y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s finished
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=1,
                   warm_start=False)
```

```
# train score
lr.score(train_vector, y_train)
```

```
0.9905003166561115
```

```
# test score
lr.score(test_vector, y_test)
```

```
0.9542772861356932
```

Training a Decision Tree Model for Document Classification

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier()
tree.fit(train_vector, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
tree.score(train_vector, y_train) # Train Accuracy
```

```
1.0
```

```
tree.score(test_vector, y_test) # Test Accuracy
```

```
0.831858407079646
```

finding 20 most important terms

```
importances = tree.feature_importances_
indices = np.argsort(importances)[::-1]

for f in range(20):
    print("%2d. %-30s %f" % (f+1,
                             [w for w, n in tfidf.vocabulary_.items()
                              if n == indices[f]],
                             importances[indices[f]]))
```

```
 1. ['graphic']                    0.099578
 2. ['christ']                     0.067288
 3. ['keith']                      0.057261
 4. ['islam']                      0.046768
 5. ['church']                     0.045765
 6. ['file']                       0.039110
 7. ['pitt']                       0.036919
 8. ['doctor']                     0.031218
 9. ['atheism']                    0.028081
10. ['faith']                      0.025263
11. ['food']                       0.021838
12. ['medic']                      0.020223
13. ['imag']                       0.017457
14. ['moral']                      0.014317
15. ['3d']                         0.011887
16. ['benedikt']                   0.011624
17. ['window']                     0.011060
18. ['methodolog']                 0.010056
19. ['algorithm']                  0.010012
20. ['wwc']                        0.009134
```

Test Sentences

```
docs = ['The outbreak was declared a global pandemic by the World Hea
        'Today, computer graphics is a core technology in digitalphot
        'Arguments for atheism range from philosophical to social and
        'The Bible is a compilation of many shorter books written at
       ]
```

```
test = tfidf.transform(docs)
```

```
lr.predict(test)
```

```
array([2, 1, 0, 3], dtype=int64)
```

```
news.target_names
```

```
['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
```

Code

```
from sklearn.datasets import fetch_20newsgroups

categories = ['alt.atheism','soc.religion.christian','comp.graphics','sci.med']

news = fetch_20newsgroups(categories=categories,
                          shuffle=False,
                          random_state=42)
X = news.data
y = news.target

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,
stratify=y, random_state=1)
```

```
import re
def preprocessor(text):
    text = re.sub('<[^>]*>', '', text) # remove <...> (tags)
    text = re.sub('[\W]+', ' ', text) # remove all non-words
    text = text.lower() # change to lower cases
    return text
```

```
import nltk
from nltk.stem.porter import PorterStemmer

porter = PorterStemmer()
def tokenizer_porter(text):
    text_tokens = nltk.word_tokenize(text)
    return [porter.stem(word) for word in text_tokens]
```

```
from nltk.corpus import stopwords
nltk.download('stopwords')
stop = stopwords.words('english')
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(preprocessor=preprocessor, # preprocessing
                        tokenizer=tokenizer_porter, # stemming
                        stop_words=stop, # removing stopwords
                        max_df=0.1,
                        min_df=10
                        )
train_vector = tfidf.fit_transform(X_train)
test_vector = tfidf.transform(X_test)
train_vector = train_vector.toarray()
```

```
test_vector = test_vector.toarray()
train_vector.shape
```

```
import numpy as np
np.set_printoptions(threshold=np.inf)
print(train_vector[0])
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(penalty="l2", verbose=1)
lr.fit(train_vector, y_train)
```

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier()
tree.fit(train_vector, y_train)
importances = tree.feature_importances_
indices = np.argsort(importances)[::-1]

for f in range(20):
    print("%2d. %-30s %f" % (f+1,
                             [w for w, n in tfidf.vocabulary_.items()
                             if n == indices[f]],
                             importances[indices[f]]))
```

```
docs = ['The outbreak was declared a global pandemic by the World Health
Organization (WHO) on 11 March.',
        'Today, computer graphics is a core technology in digitalphotography,
film, video games, cell phone and computer displays,and many specialized
applications.',
        'Arguments for atheism range from philosophical to social and historical
approaches.',
        'The Bible is a compilation of many shorter books written at different
times by a variety of authors, and later assembled into the biblical canon.'
        ]
test = tfidf.transform(docs)
```

Result(Captured images)

```
X[0]
```

✓ 0.4s Python

```
"From: keith@cco.caltech.edu (Keith Allan Schneider)\nSubject: Re: <Political
Atheists?\nOrganization: California Institute of Technology, Pasadena\nLines:
14\nNNTP-Posting-Host: lloyd.caltech.edu\n\nbobbe@vice.ICO.TEK.COM (Robert
Beauchaine) writes:\n\n>To show that the examples I and others\n>have provided are
*not* counter examples of your supposed inherent\n>moral hypothesis, you have to
successfully argue that\n>domestication removes or alters this morality.\n\nI think
that domestication will change behavior to a large degree.\nDomesticated animals
exhibit behaviors not found in the wild.  I\ndon't think that they can be viewed as
good representatives of the\nwild animal kingdom, since they have been bred for
thousands of years\nto produce certain behaviors, etc.\n\nkeith\n"
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(penalty="l2", verbose=1)
lr.fit(train_vector, y_train)
```

✓ 1.4s Python

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.9s finished

LogisticRegression(verbose=1)
```

```
lr.score(train_vector,y_train)
```

✓ 0.6s Python

```
0.9911336288790373
```

```
lr.score(test_vector, y_test)
```

✓ 0.7s Python

```
0.9542772861356932
```

```python
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier()
tree.fit(train_vector, y_train)
```

✓ 1.3s Python

```
DecisionTreeClassifier()
```

```python
tree.score(train_vector,y_train)
```

✓ 0.7s Python

```
1.0
```

```python
tree.score(test_vector,y_test)
```

✓ 0.6s Python

```
0.8407079646017699
```

```python
importances = tree.feature_importances_
indices = np.argsort(importances)[::-1]

for f in range(20):
    print("%2d. %-30s %f" % (f+1,
                             [w for w, n in tfidf.vocabulary_.items()
                             if n == indices[f]],
                             importances[indices[f]]))
```

✓ 0.1s Python

```
 1. ['graphic']                    0.099578
 2. ['christ']                     0.067288
 3. ['keith']                      0.057261
 4. ['islam']                      0.046768
 5. ['church']                     0.045765
 6. ['file']                       0.039110
 7. ['pitt']                       0.035279
 8. ['doctor']                     0.030089
 9. ['atheism']                    0.028081
10. ['faith']                      0.025263
11. ['food']                       0.021838
12. ['medic']                      0.020223
13. ['imag']                       0.017457
14. ['moral']                      0.014317
15. ['3d']                         0.011887
16. ['benedikt']                   0.011624
17. ['window']                     0.011060
18. ['methodolog']                 0.010056
19. ['algorithm']                  0.010012
20. ['saturn']                     0.009134
```

```
docs = ['The outbreak was declared a global pandemic by the World Health Organizati
        'Today, computer graphics is a core technology in digitalphotography, film,
        'Arguments for atheism range from philosophical to social and historical ap
        'The Bible is a compilation of many shorter books written at different time
        ]
```

✓ 0.1s Python

```
test = tfidf.transform(docs)
```

✓ 0.1s Python

```
lr.predict(test)
```

✓ 0.7s Python

```
array([2, 1, 0, 3], dtype=int64)
```

```
news.target_names
```

✓ 0.5s Python

```
['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
```

Description

*preprocessor 를 직접 생성하고 PorterStemmer()를 사용하여 Porter stemmer tokenizer 를 생성한다. TfidfVectorizer()를 설정하는데 preprocessor 를 직접 생성한 preprocessor 로 설정하고 tokenizer 를 PorterStemmer()를 사용하여 만든 tokenizer 로 설정하고 stop_words 를 stop 으로 설정하고 "*Drop terms occurred in more than 10% of docs"*를 만족시키기 위해 max_df=0.1 로 설정하고, "*Drop terms occurred in less than 10 docs"*를 만족시키기 위해 min_df=10 으로 설정한다. 이후 X_train 을 fit_transform 해서 train_vector 를 생성하고 X_test 를 transform 해서 test_vector 를 생성한다. 이후 LogisticRegression()과 DecisionTreeClassifier()를 통해 학습시키고 정확도를 측정한다. 또한 중요도를 측정한다.*

Note

1. Summit the file to e-class as pdf.
2. Specify your file name as "hw3_<StudentID>_<Name>.pdf"