Machine Learning

August 6, 2023

1 Machine Learning

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

[2]:
```
%matplotlib inline
```

1.1 Loading Data

[3]:
```
ecg_data = pd.read_csv("./processed_data/ecg_processed_data.csv", index_col=0)
```

[4]:
```
ecg_data.head()
```

[4]:
```
   Record ID  Segment Start  Segment End         0         1         2  \
0        100             13          213  0.059449  0.055308  0.052035
1        100            307          507  0.084239  0.087566  0.088956
2        100            563          763 -0.006937 -0.005690 -0.005184
3        100            883         1083  0.072086  0.073128  0.073026
4        100           1168         1368  0.084762  0.083995  0.082327

          3         4         5         6  …       191       192       193  \
0  0.049666  0.047961  0.046922  0.046801  …  0.029812  0.031902  0.033233
1  0.088621  0.086919  0.084459  0.081823  …  0.020380  0.020170  0.020049
2 -0.005172 -0.005405 -0.005802 -0.006385  … -0.083923 -0.084476 -0.084798
3  0.071802  0.069731  0.067336  0.065156  …  0.040917  0.040820  0.040605
4  0.080127  0.077701  0.075216  0.072963  …  0.024691  0.025225  0.025977

        194       195       196       197       198       199  \
0  0.033981  0.034276  0.034166  0.033702  0.033134  0.032648
1  0.020272  0.020719  0.020971  0.020866  0.020622  0.020434
2 -0.084841 -0.084325 -0.082953 -0.080644 -0.077424 -0.073181
3  0.040177  0.039328  0.038051  0.036671  0.035565  0.034817
4  0.027249  0.029211  0.031715  0.034364  0.036462  0.037443

  Annotation Class
```

```
0                 N
1                 N
2                 N
3                 N
4                 N

[5 rows x 204 columns]
```

[5]:
```python
ecg_data["Annotation Class"].unique()
```

[5]:
```
array(['N', 'V', '/', 'L', 'R'], dtype=object)
```

[6]:
```python
ecg_data["Annotation Class"].nunique()
```

[6]:
```
5
```

[7]:
```python
ecg_data["Annotation Class"].value_counts()
```

[7]:
```
Annotation Class
N    73439
L     8068
R     7255
V     6793
/     3619
Name: count, dtype: int64
```

1.2 Preprocessing

1.2.1 Converting Non Numeric Column to Numeric discontinuous Columns

[8]:
```python
annotation_dict = dict()
for i, symbol in enumerate(ecg_data["Annotation Class"].unique()):
    annotation_dict[symbol] = i + 1
ecg_data["Annotation Class Numeric"] = ecg_data["Annotation Class"].apply(
    lambda x: annotation_dict[x]
)
```

[9]:
```python
ecg_data.head()
```

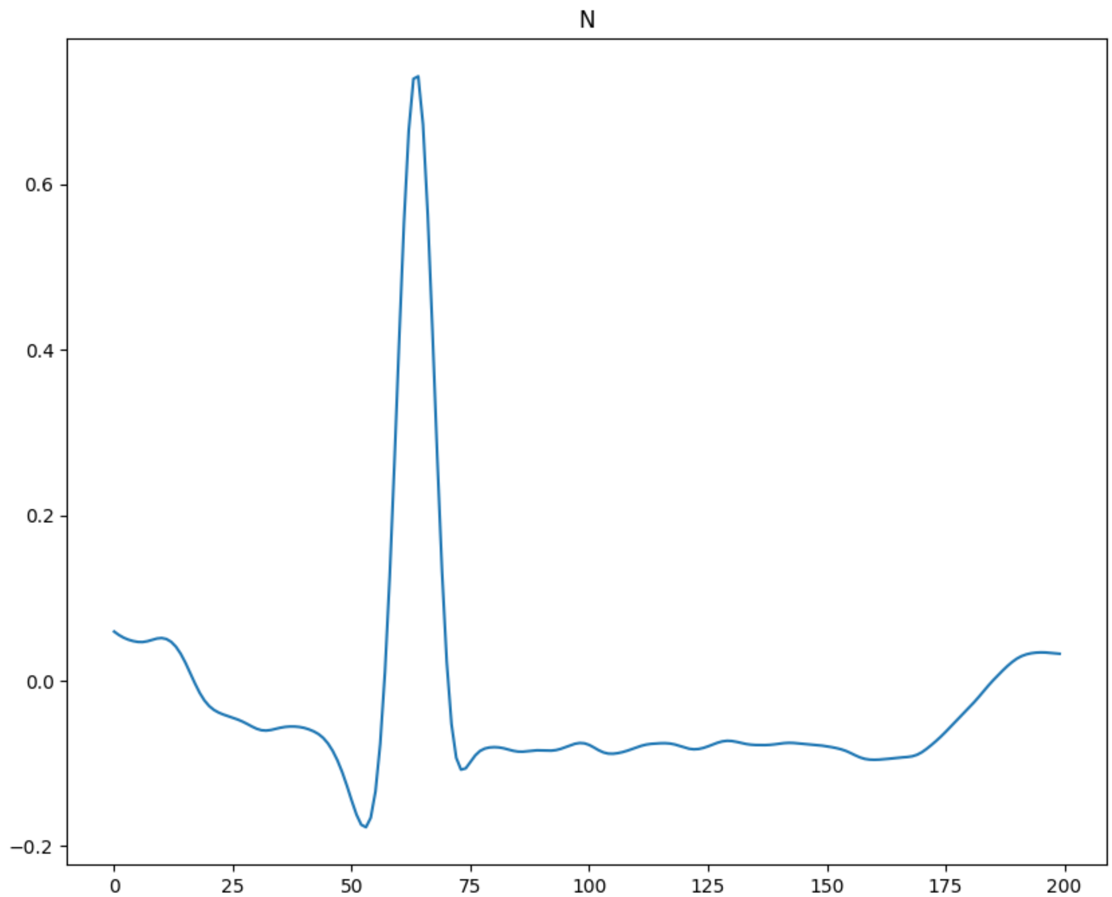
[9]:
```
   Record ID  Segment Start  Segment End         0         1         2  \
0        100             13          213  0.059449  0.055308  0.052035
1        100            307          507  0.084239  0.087566  0.088956
2        100            563          763 -0.006937 -0.005690 -0.005184
3        100            883         1083  0.072086  0.073128  0.073026
4        100           1168         1368  0.084762  0.083995  0.082327

          3         4         5         6  …       192       193       194  \
```

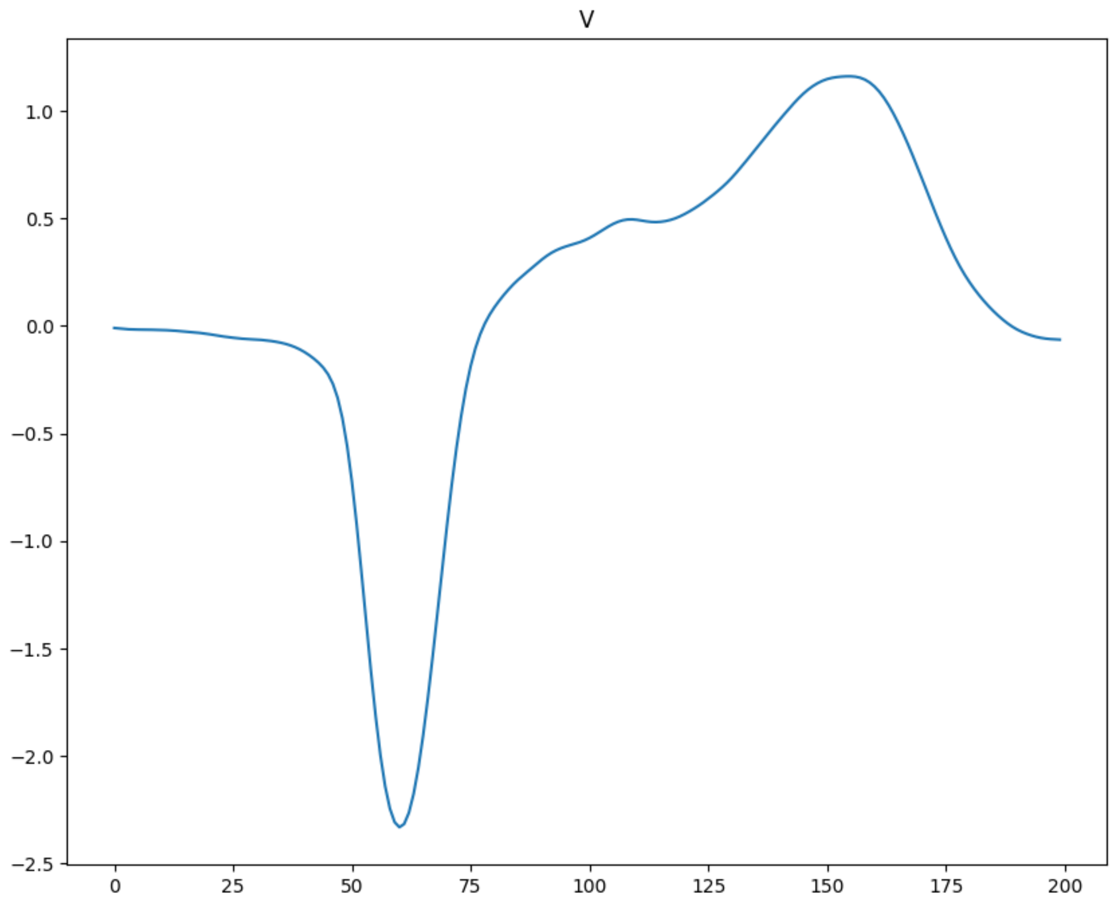
```
0  0.049666  0.047961  0.046922  0.046801  …  0.031902  0.033233  0.033981
1  0.088621  0.086919  0.084459  0.081823  …  0.020170  0.020049  0.020272
2 -0.005172 -0.005405 -0.005802 -0.006385  … -0.084476 -0.084798 -0.084841
3  0.071802  0.069731  0.067336  0.065156  …  0.040820  0.040605  0.040177
4  0.080127  0.077701  0.075216  0.072963  …  0.025225  0.025977  0.027249

        195       196       197       198       199  Annotation Class  \
0  0.034276  0.034166  0.033702  0.033134  0.032648                 N
1  0.020719  0.020971  0.020866  0.020622  0.020434                 N
2 -0.084325 -0.082953 -0.080644 -0.077424 -0.073181                 N
3  0.039328  0.038051  0.036671  0.035565  0.034817                 N
4  0.029211  0.031715  0.034364  0.036462  0.037443                 N

   Annotation Class Numeric
0                         1
1                         1
2                         1
3                         1
4                         1

[5 rows x 205 columns]
```

[10]:
```
x_columns = [str(i) for i in range(0, 200)]
```

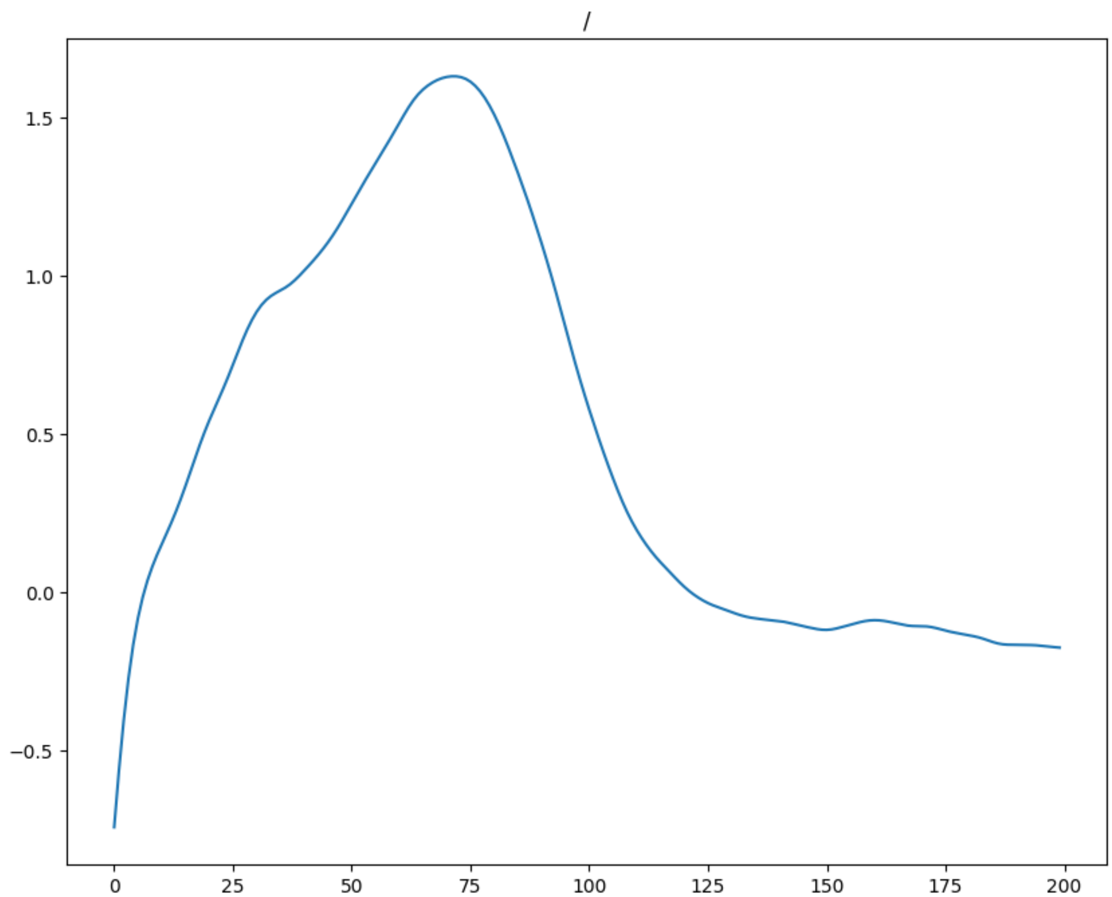
1.3 What each Annotation looks like?

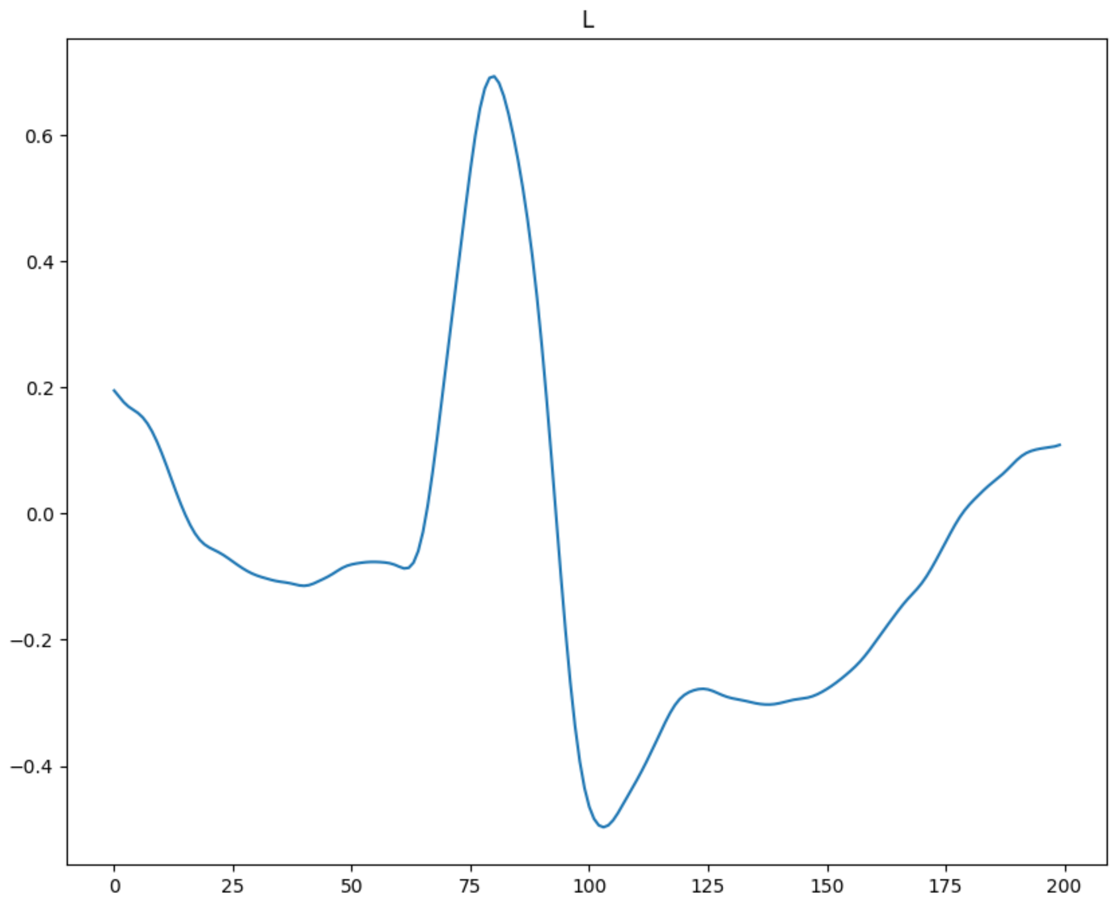
[11]:
```
for symbol in ecg_data["Annotation Class"].unique():
    readings = (
        ecg_data[ecg_data["Annotation Class"] == symbol].head(1)[x_columns].
 ↪values[0]
    )
    plt.figure(figsize=(10, 8))
    plt.title(label=symbol)
    plt.plot(readings)
    plt.show()
```

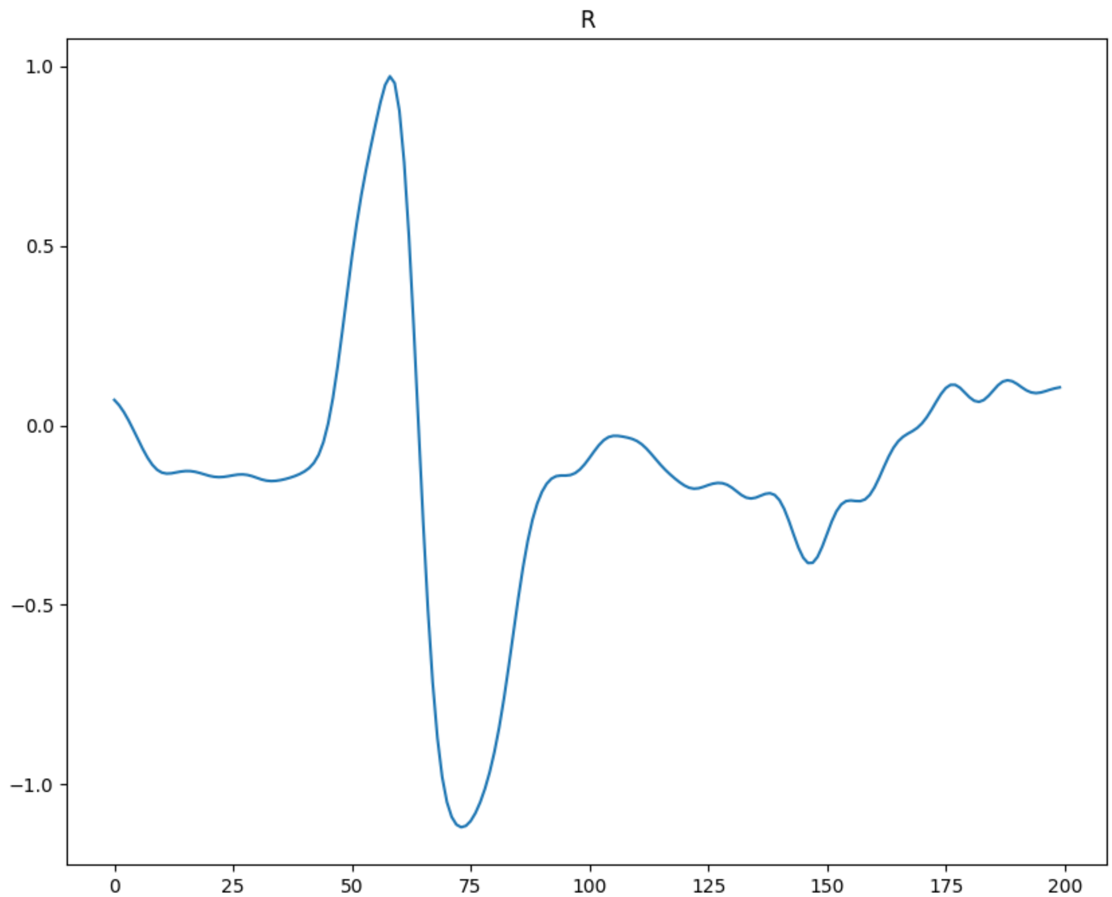

N
0.6
0.4
0.2
0.0
−0.2
0
25
50
75
100
125
150
175
200



V
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
0
25
50
75
100
125
150
175
200


/
1.5
1.0
0.5
0.0
−0.5
0
25
50
75
100
125
150
175
200


L
0.6
0.4
0.2
0.0
−0.2
−0.4
0
25
50
75
100
125
150
175
200



1.4 Splitting into Test and Train Data

[12]:
```
from sklearn.preprocessing import MinMaxScaler
```

[13]:
```
from sklearn.pipeline import Pipeline
```

[14]:
```
from sklearn.model_selection import train_test_split
```

[15]:
```
X = ecg_data[x_columns].values
```

[16]:
```
y = ecg_data["Annotation Class Numeric"].values
```

[17]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=101, stratify=y, shuffle=True
)
X_train_1, X_val, y_train_1, y_val = train_test_split(
    X_train, y_train, test_size=0.2, stratify=y_train, shuffle=True,␣
 ↪random_state=101
```

```
)
```

[18]:
```
X_train.shape
```

[18]: (79339, 200)

[19]:
```
y_train.shape
```

[19]: (79339,)

[20]:
```
X_test.shape
```

[20]: (19835, 200)

[21]:
```
y_test.shape
```

[21]: (19835,)

1.5 Logistic Regression

[22]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

[23]:
```
from skopt import BayesSearchCV
```

[24]:
```
lr_params = {"multi_class": ["ovr", "multinomial", "auto"]}
```

[25]:
```
pipeline = Pipeline(
    [
        ("minmax", MinMaxScaler()),
        ("lr", GridSearchCV(LogisticRegression(), lr_params, verbose=3)),
    ]
)
```

[26]:
```
pipeline.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
```

```
/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
[CV 1/5] END …multi_class=ovr;, score=0.839 total time=  11.4s
/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
[CV 2/5] END …multi_class=ovr;, score=0.843 total time=  13.2s
/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
[CV 3/5] END …multi_class=ovr;, score=0.843 total time=  10.8s
/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
[CV 4/5] END …multi_class=ovr;, score=0.845 total time=  11.3s
/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
[CV 5/5] END …multi_class=ovr;, score=0.839 total time=  12.8s
```

```
/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
[CV 1/5] END …multi_class=multinomial;, score=0.840 total time=  16.8s
```

```
/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
[CV 2/5] END …multi_class=multinomial;, score=0.842 total time=   9.1s
```

```
/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
[CV 3/5] END …multi_class=multinomial;, score=0.840 total time=  11.6s
```

```
/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

[CV 4/5] END …multi_class=multinomial;, score=0.838 total time=  10.5s

/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

[CV 5/5] END …multi_class=multinomial;, score=0.832 total time=  11.0s

/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

[CV 1/5] END …multi_class=auto;, score=0.840 total time=  16.4s

/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(

[CV 2/5] END …multi_class=auto;, score=0.842 total time=  14.3s

/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

[CV 3/5] END …multi_class=auto;, score=0.840 total time=  11.4s

/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

[CV 4/5] END …multi_class=auto;, score=0.838 total time=  12.4s

/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

[CV 5/5] END …multi_class=auto;, score=0.832 total time=  12.7s

/home/alton/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[26]:
```
Pipeline(steps=[('minmax', MinMaxScaler()),
                ('lr',
                 GridSearchCV(estimator=LogisticRegression(),
                              param_grid={'multi_class': ['ovr', 'multinomial',
                                                          'auto']},
                              verbose=3))])
```

[27]:
```python
pred = pipeline.predict(X_test)
```

[28]:
```python
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
```

[29]:
```python
print(confusion_matrix(y_test, pred))
```

```
[[14562    27     1    66    32]
 [  647   476    22   112   101]
 [    6     3   703    12     0]
 [ 1051    26     1   534     2]
 [  981    14     0    23   433]]
```

[30]:
```python
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           1       0.84      0.99      0.91     14688
           2       0.87      0.35      0.50      1358
           3       0.97      0.97      0.97       724
           4       0.71      0.33      0.45      1614
           5       0.76      0.30      0.43      1451

    accuracy                           0.84     19835
   macro avg       0.83      0.59      0.65     19835
weighted avg       0.83      0.84      0.81     19835
```

1.6 KNN

[31]:
```python
from sklearn.neighbors import KNeighborsClassifier
```

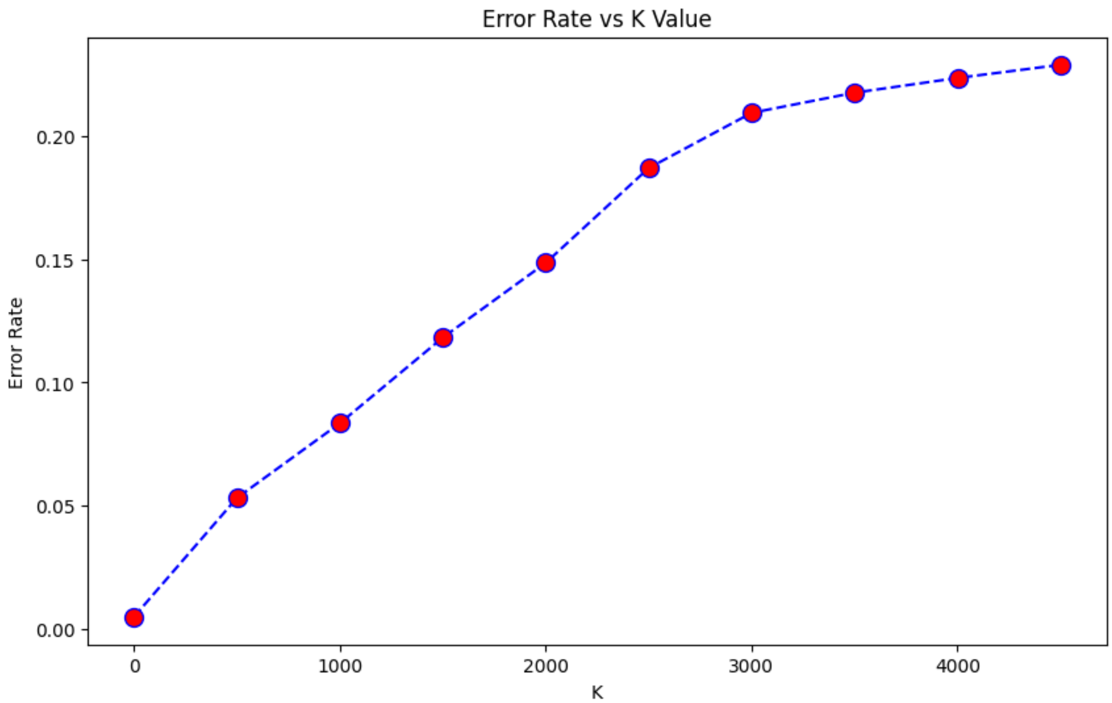
[32]:
```python
error_rate = []
for i in range(1, 5000, 500):
```

```
    knn = Pipeline(
        [("minmax", MinMaxScaler()), ("knn",␣
↪KNeighborsClassifier(n_neighbors=i))]
    )
    knn.fit(X_train_1, y_train_1)
    pred_i = knn.predict(X_val)
    print(f"Neighbours {i}")
    print(f"Loss: {np.mean(pred_i !=y_val)}")
    # Take the mean where prediction is not equal to actual
    error_rate.append(np.mean(pred_i != y_val))
```

```
Neighbours 1
Loss: 0.004726493571968742
Neighbours 501
Loss: 0.053062767834635746
Neighbours 1001
Loss: 0.08343836652382153
Neighbours 1501
Loss: 0.11828837912780439
Neighbours 2001
Loss: 0.14866397781699017
Neighbours 2501
Loss: 0.1872321653642551
Neighbours 3001
Loss: 0.2096672548525334
Neighbours 3501
Loss: 0.21785984371061257
Neighbours 4001
Loss: 0.22384673556843962
Neighbours 4501
Loss: 0.22920342828333753
```

[33]:
```
plt.figure(figsize=(10, 6))
plt.plot(
    range(1, 5000, 500),
    error_rate,
    color="blue",
    linestyle="dashed",
    marker="o",
    markerfacecolor="red",
    markersize=10,
)
plt.title("Error Rate vs K Value")
plt.xlabel("K")
plt.ylabel("Error Rate")
```

[33]:
```
Text(0, 0.5, 'Error Rate')
```



[34]:
```
knn = Pipeline(
    [("minmax", MinMaxScaler()), ("knn", KNeighborsClassifier(n_neighbors=5))]
)
knn.fit(X_train_1, y_train_1)
```

[34]:
```
Pipeline(steps=[('minmax', MinMaxScaler()), ('knn', KNeighborsClassifier())])
```

[35]:
```
pred = knn.predict(X_test)
```

[36]:
```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00     14688
           2       0.98      0.95      0.96      1358
           3       1.00      1.00      1.00       724
           4       0.99      0.99      0.99      1614
           5       0.99      1.00      0.99      1451

    accuracy                           0.99     19835
   macro avg       0.99      0.99      0.99     19835
weighted avg       0.99      0.99      0.99     19835
```

[37]:
```
print(confusion_matrix(y_test, pred))
```

```
[[14659    20     2     3     4]
 [   47  1288     1    18     4]
 [    0     0   723     1     0]
 [    7     4     0  1603     0]
 [    6     0     0     1  1444]]
```

[38]:
```
print(accuracy_score(y_test, pred))
```

```
0.9940509200907487
```

1.7 Decision Tree

[39]:
```
from sklearn.tree import DecisionTreeClassifier
```

[40]:
```
dt_params = {
    "criterion": ["gini", "entropy", "log_loss"],
    "splitter": ["best", "random"],
}
dt = Pipeline(
    [
        ("minmax", MinMaxScaler()),
        ("Bayes", GridSearchCV(DecisionTreeClassifier(), dt_params, verbose=3)),
    ]
)
```

[41]:
```
dt.fit(X_train_1, y_train_1)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[CV 1/5] END …criterion=gini, splitter=best;, score=0.979 total time=  45.1s
[CV 2/5] END …criterion=gini, splitter=best;, score=0.978 total time=  42.1s
[CV 3/5] END …criterion=gini, splitter=best;, score=0.979 total time=  39.7s
[CV 4/5] END …criterion=gini, splitter=best;, score=0.978 total time=  43.6s
[CV 5/5] END …criterion=gini, splitter=best;, score=0.979 total time=  40.9s
[CV 1/5] END …criterion=gini, splitter=random;, score=0.982 total time=   3.7s
[CV 2/5] END …criterion=gini, splitter=random;, score=0.980 total time=   4.4s
[CV 3/5] END …criterion=gini, splitter=random;, score=0.981 total time=   2.9s
[CV 4/5] END …criterion=gini, splitter=random;, score=0.983 total time=   3.5s
[CV 5/5] END …criterion=gini, splitter=random;, score=0.984 total time=   3.1s
[CV 1/5] END ..criterion=entropy, splitter=best;, score=0.984 total time=  34.7s
[CV 2/5] END ..criterion=entropy, splitter=best;, score=0.981 total time=  34.1s
[CV 3/5] END ..criterion=entropy, splitter=best;, score=0.983 total time=  33.4s
[CV 4/5] END ..criterion=entropy, splitter=best;, score=0.983 total time=  33.9s
[CV 5/5] END ..criterion=entropy, splitter=best;, score=0.981 total time=  33.3s
[CV 1/5] END criterion=entropy, splitter=random;, score=0.984 total time=   2.0s
[CV 2/5] END criterion=entropy, splitter=random;, score=0.983 total time=   2.3s
[CV 3/5] END criterion=entropy, splitter=random;, score=0.984 total time=   2.3s
```

```
[CV 4/5] END criterion=entropy, splitter=random;, score=0.983 total time=   1.9s
[CV 5/5] END criterion=entropy, splitter=random;, score=0.983 total time=   2.1s
[CV 1/5] END .criterion=log_loss, splitter=best;, score=0.983 total time=  34.7s
[CV 2/5] END .criterion=log_loss, splitter=best;, score=0.982 total time=  33.6s
[CV 3/5] END .criterion=log_loss, splitter=best;, score=0.984 total time=  33.3s
[CV 4/5] END .criterion=log_loss, splitter=best;, score=0.983 total time=  35.3s
[CV 5/5] END .criterion=log_loss, splitter=best;, score=0.981 total time=  38.4s
[CV 1/5] END criterion=log_loss, splitter=random;, score=0.984 total time=
2.4s
[CV 2/5] END criterion=log_loss, splitter=random;, score=0.982 total time=
2.6s
[CV 3/5] END criterion=log_loss, splitter=random;, score=0.984 total time=
2.5s
[CV 4/5] END criterion=log_loss, splitter=random;, score=0.983 total time=
2.1s
[CV 5/5] END criterion=log_loss, splitter=random;, score=0.983 total time=
1.9s
```

[41]:
```
Pipeline(steps=[('minmax', MinMaxScaler()),
                ('Bayes',
                 GridSearchCV(estimator=DecisionTreeClassifier(),
                              param_grid={'criterion': ['gini', 'entropy',
                                                        'log_loss'],
                                          'splitter': ['best', 'random']},
                              verbose=3))])
```

[42]:
```
pred_val = dt.predict(X_val)
```

[43]:
```
print(confusion_matrix(y_val, pred_val))
```

```
[[11663    40     1    24    22]
 [   52  1008     7    13     7]
 [    2     6   569     2     0]
 [   25    15     0  1250     1]
 [   20    10     0     2  1129]]
```

[44]:
```
print(classification_report(y_val, pred_val))
```

```
              precision    recall  f1-score   support

           1       0.99      0.99      0.99     11750
           2       0.93      0.93      0.93      1087
           3       0.99      0.98      0.98       579
           4       0.97      0.97      0.97      1291
           5       0.97      0.97      0.97      1161

    accuracy                           0.98     15868
   macro avg       0.97      0.97      0.97     15868
```

```
weighted avg       0.98      0.98      0.98     15868
```

[45]:
```python
pred = dt.predict(X_test)
```

[46]:
```python
print(confusion_matrix(y_test, pred))
```

```
[[14585    60     2    21    20]
 [   66  1255     5    16    16]
 [    4     4   715     1     0]
 [   22    17     0  1570     5]
 [   27     3     0     2  1419]]
```

[47]:
```python
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           1       0.99      0.99      0.99     14688
           2       0.94      0.92      0.93      1358
           3       0.99      0.99      0.99       724
           4       0.98      0.97      0.97      1614
           5       0.97      0.98      0.97      1451

    accuracy                           0.99     19835
   macro avg       0.97      0.97      0.97     19835
weighted avg       0.99      0.99      0.99     19835
```

1.8 Random Forest

[48]:
```python
from sklearn.ensemble import RandomForestClassifier
```

[49]:
```python
rand = Pipeline(
    [("minmax", MinMaxScaler()), ("rand",␣
 ↪RandomForestClassifier(criterion="gini"))]
)
```

[50]:
```python
rand.fit(X_train_1, y_train_1)
```

[50]:
```
Pipeline(steps=[('minmax', MinMaxScaler()), ('rand', RandomForestClassifier())])
```

[51]:
```python
pred_val = rand.predict(X_val)
```

[52]:
```python
print(classification_report(y_val, pred_val))
```

```
              precision    recall  f1-score   support

           1       0.99      1.00      1.00     11750
```

```
           2       0.97      0.96      0.96      1087
           3       1.00      0.99      0.99       579
           4       0.99      0.97      0.98      1291
           5       1.00      0.98      0.99      1161

    accuracy                           0.99     15868
   macro avg       0.99      0.98      0.99     15868
weighted avg       0.99      0.99      0.99     15868
```

[53]:
```python
print(confusion_matrix(y_val, pred_val))
```

```
[[11731    18     0     1     0]
 [   38  1041     2     5     1]
 [    2     2   574     1     0]
 [   23    10     0  1258     0]
 [   24     1     0     0  1136]]
```

[54]:
```python
pred = rand.predict(X_test)
```

[55]:
```python
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           1       0.99      1.00      1.00     14688
           2       0.98      0.95      0.96      1358
           3       1.00      1.00      1.00       724
           4       0.99      0.98      0.99      1614
           5       1.00      0.99      0.99      1451

    accuracy                           0.99     19835
   macro avg       0.99      0.98      0.99     19835
weighted avg       0.99      0.99      0.99     19835
```

[56]:
```python
print(confusion_matrix(y_test, pred))
```

```
[[14667    15     1     3     2]
 [   59  1289     2     5     3]
 [    2     1   721     0     0]
 [   19    14     0  1581     0]
 [   16     1     0     2  1432]]
```

1.9 SVM

[57]:
```python
from sklearn.svm import SVC
```

[58]:
```
svm_param_grid = {"C": [0.1, 1, 10, 100, 1000], "gamma": [1, 0.1, 0.01, 0.001,␣
↪0.0001]}
```

[59]:
```
svm = Pipeline([("minmax", MinMaxScaler()), ("SVM", SVC(C=0.1, gamma=1))])
```

[ ]:
```

```

[60]:
```
svm.fit(X_train_1, y_train_1)
```

[60]:
```
Pipeline(steps=[('minmax', MinMaxScaler()), ('SVM', SVC(C=0.1, gamma=1))])
```

[61]:
```
pred_val = svm.predict(X_val)
```

[62]:
```
print(classification_report(y_val, pred_val))
```

```
              precision    recall  f1-score   support

           1       0.98      0.99      0.99     11750
           2       0.95      0.92      0.94      1087
           3       0.99      0.99      0.99       579
           4       0.97      0.93      0.95      1291
           5       0.99      0.91      0.95      1161

    accuracy                           0.98     15868
   macro avg       0.98      0.95      0.96     15868
weighted avg       0.98      0.98      0.98     15868
```

[63]:
```
print(confusion_matrix(y_val, pred_val))
```

```
[[11687    34     0    21     8]
 [   73   996     5    11     2]
 [    1     3   574     1     0]
 [   83     9     0  1199     0]
 [   98     1     0     0  1062]]
```

[64]:
```
pred = svm.predict(X_test)
```

[65]:
```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           1       0.98      0.99      0.99     14688
           2       0.95      0.90      0.93      1358
           3       1.00      1.00      1.00       724
           4       0.97      0.93      0.95      1614
           5       0.98      0.92      0.95      1451
```

```
    accuracy                           0.98     19835
   macro avg       0.98      0.95      0.96     19835
weighted avg       0.98      0.98      0.98     19835
```

[66]:
```python
print(confusion_matrix(y_test, pred))
```

```
[[14608    39     0    32     9]
 [  102  1228     2    14    12]
 [    1     1   722     0     0]
 [   95    17     0  1502     0]
 [  118     2     0     1  1330]]
```

1.10 Gaussian Naive Bayes

[67]:
```python
from sklearn.naive_bayes import GaussianNB
```

[68]:
```python
gnb = Pipeline([("minmax", MinMaxScaler()), ("GNB", GaussianNB())])
```

[69]:
```python
gnb.fit(X_train_1, y_train_1)
```

[69]:
```
Pipeline(steps=[('minmax', MinMaxScaler()), ('GNB', GaussianNB())])
```

[70]:
```python
pred_val = gnb.predict(X_val)
```

[71]:
```python
print(classification_report(y_val, pred_val))
```

```
              precision    recall  f1-score   support

           1       0.94      0.69      0.79     11750
           2       0.44      0.56      0.50      1087
           3       0.63      0.93      0.75       579
           4       0.29      0.79      0.42      1291
           5       0.54      0.72      0.62      1161

    accuracy                           0.70     15868
   macro avg       0.57      0.74      0.62     15868
weighted avg       0.82      0.70      0.73     15868
```

[72]:
```python
print(confusion_matrix(y_val, pred_val))
```

```
[[8065  649  143 2232  661]
 [  23  612  173  236   43]
 [   0   39  540    0    0]
 [ 188   71    5 1021    6]
 [ 265    8    0   56  832]]
```

[73]:
```
pred = gnb.predict(X_test)
```

[74]:
```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           1       0.95      0.70      0.81     14688
           2       0.44      0.55      0.49      1358
           3       0.67      0.94      0.78       724
           4       0.30      0.80      0.43      1614
           5       0.56      0.72      0.63      1451

    accuracy                           0.71     19835
   macro avg       0.58      0.74      0.63     19835
weighted avg       0.82      0.71      0.74     19835
```

[75]:
```
print(confusion_matrix(y_test, pred))
```

```
[[10325   807   149  2660   747]
 [   38   748   174   324    74]
 [    1    44   678     1     0]
 [  217    91     6  1291     9]
 [  341    13     0    53  1044]]
```

2 ANN

[76]:
```
from tensorflow.keras.models import Sequential
```

```
2023-07-05 10:24:47.054506: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-07-05 10:24:47.210747: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-07-05 10:24:47.212063: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2023-07-05 10:24:49.061011: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

[77]:
```
from tensorflow.keras.layers import Dense, Dropout
```

[78]:
```
from tensorflow.keras.callbacks import EarlyStopping
```

[79]:
```
from keras.utils import to_categorical
```

[80]:
```
early_stop = EarlyStopping(monitor="val_loss", mode="min", verbose=1,␣
↪patience=25)
```

[81]:
```
model = Sequential()
```

[82]:
```
model.add(Dense(300, activation="relu"))
```

[83]:
```
model.add(Dropout(0.5))
```

[84]:
```
model.add(Dense(150, activation="relu"))
```

[85]:
```
model.add(Dropout(0.5))
```

[86]:
```
model.add(Dense(75, activation="relu"))
```

[87]:
```
model.add(Dropout(0.5))
```

[88]:
```
model.add(Dense(35, activation="relu"))
```

[89]:
```
model.add(Dropout(0.5))
```

[90]:
```
model.add(Dense(15, activation="relu"))
```

[91]:
```
model.add(Dropout(0.5))
```

[92]:
```
model.add(Dense(5, activation="softmax"))
```

[93]:
```
model.compile(loss="categorical_crossentropy", optimizer="adam")
```

[94]:
```
model.fit(
    x=X_train_1,
    y=to_categorical(y_train_1)[:, 1:],
    epochs=600,
    validation_data=(X_val, to_categorical(y_val)[:, 1:]),
    batch_size=32,
    callbacks=[early_stop],
)
```

```
Epoch 1/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.6362 -
val_loss: 0.3861
Epoch 2/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.4144 -
val_loss: 0.2850
Epoch 3/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.3508 -
val_loss: 0.2124
```

```
Epoch 4/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.3092 -
val_loss: 0.1773
Epoch 5/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.2719 -
val_loss: 0.1573
Epoch 6/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.2562 -
val_loss: 0.1316
Epoch 7/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.2406 -
val_loss: 0.1300
Epoch 8/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.2297 -
val_loss: 0.1213
Epoch 9/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.2280 -
val_loss: 0.1042
Epoch 10/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.2090 -
val_loss: 0.1170
Epoch 11/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.2083 -
val_loss: 0.1070
Epoch 12/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.2004 -
val_loss: 0.0979
Epoch 13/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1961 -
val_loss: 0.1079
Epoch 14/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1876 -
val_loss: 0.0866
Epoch 15/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1791 -
val_loss: 0.0859
Epoch 16/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1778 -
val_loss: 0.0813
Epoch 17/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1747 -
val_loss: 0.0818
Epoch 18/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1743 -
val_loss: 0.0816
Epoch 19/600
1984/1984 [==============================] - 12s 6ms/step - loss: 0.1716 -
val_loss: 0.0844
```

```
Epoch 20/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1668 -
val_loss: 0.0889
Epoch 21/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1635 -
val_loss: 0.0842
Epoch 22/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1624 -
val_loss: 0.0920
Epoch 23/600
1984/1984 [==============================] - 12s 6ms/step - loss: 0.1621 -
val_loss: 0.0810
Epoch 24/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1637 -
val_loss: 0.0872
Epoch 25/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1607 -
val_loss: 0.0878
Epoch 26/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1612 -
val_loss: 0.0821
Epoch 27/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1553 -
val_loss: 0.0785
Epoch 28/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.1531 -
val_loss: 0.0749
Epoch 29/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1549 -
val_loss: 0.0856
Epoch 30/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.1564 -
val_loss: 0.0721
Epoch 31/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1503 -
val_loss: 0.0797
Epoch 32/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.1552 -
val_loss: 0.0732
Epoch 33/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1500 -
val_loss: 0.0739
Epoch 34/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1444 -
val_loss: 0.0682
Epoch 35/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.1547 -
val_loss: 0.0729
```

```
Epoch 36/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1491 -
val_loss: 0.0668
Epoch 37/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1497 -
val_loss: 0.0641
Epoch 38/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.1439 -
val_loss: 0.0710
Epoch 39/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1506 -
val_loss: 0.0684
Epoch 40/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.1373 -
val_loss: 0.0713
Epoch 41/600
1984/1984 [==============================] - 11s 6ms/step - loss: 0.1434 -
val_loss: 0.0677
Epoch 42/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.1417 -
val_loss: 0.0767
Epoch 43/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.1415 -
val_loss: 0.0708
Epoch 44/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1507 -
val_loss: 0.0650
Epoch 45/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1439 -
val_loss: 0.0724
Epoch 46/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.1487 -
val_loss: 0.0691
Epoch 47/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1398 -
val_loss: 0.0787
Epoch 48/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1477 -
val_loss: 0.0849
Epoch 49/600
1984/1984 [==============================] - 10s 5ms/step - loss: 0.1405 -
val_loss: 0.0660
Epoch 50/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1400 -
val_loss: 0.0718
Epoch 51/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1519 -
val_loss: 0.0691
```

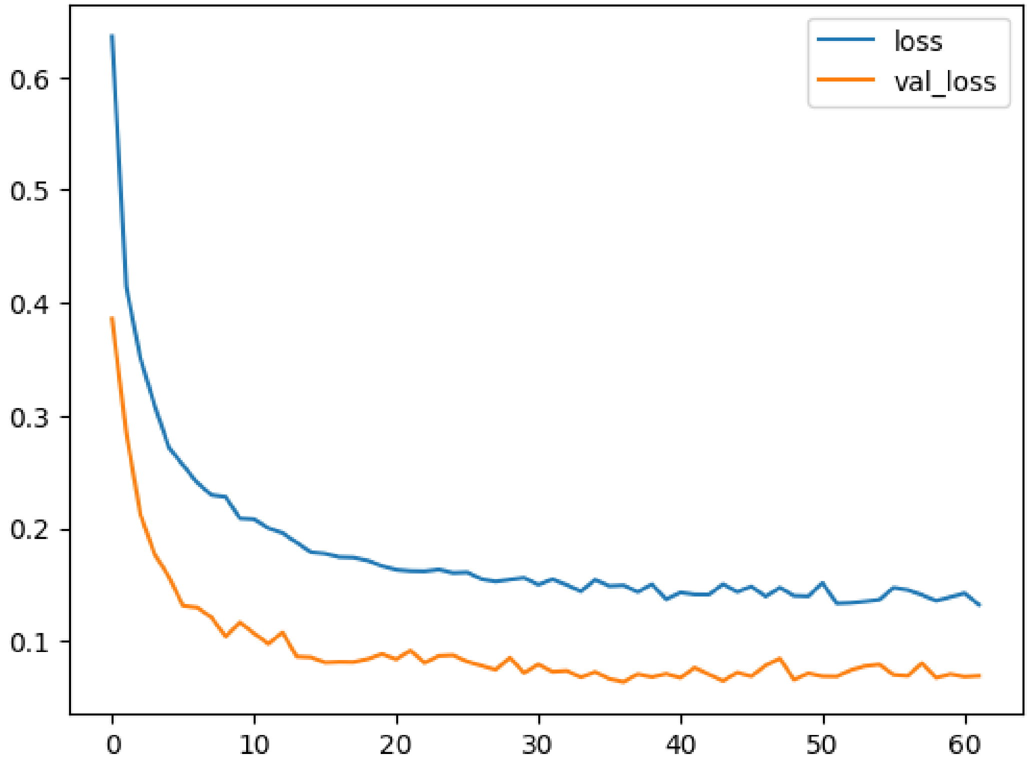
```
Epoch 52/600
1984/1984 [==============================] - 11s 5ms/step - loss: 0.1337 -
val_loss: 0.0689
Epoch 53/600
1984/1984 [==============================] - 384s 194ms/step - loss: 0.1342 -
val_loss: 0.0746
Epoch 54/600
1984/1984 [==============================] - 9s 4ms/step - loss: 0.1354 -
val_loss: 0.0782
Epoch 55/600
1984/1984 [==============================] - 8s 4ms/step - loss: 0.1369 -
val_loss: 0.0796
Epoch 56/600
1984/1984 [==============================] - 8s 4ms/step - loss: 0.1476 -
val_loss: 0.0704
Epoch 57/600
1984/1984 [==============================] - 8s 4ms/step - loss: 0.1456 -
val_loss: 0.0695
Epoch 58/600
1984/1984 [==============================] - 8s 4ms/step - loss: 0.1412 -
val_loss: 0.0807
Epoch 59/600
1984/1984 [==============================] - 8s 4ms/step - loss: 0.1359 -
val_loss: 0.0677
Epoch 60/600
1984/1984 [==============================] - 8s 4ms/step - loss: 0.1393 -
val_loss: 0.0711
Epoch 61/600
1984/1984 [==============================] - 9s 4ms/step - loss: 0.1427 -
val_loss: 0.0687
Epoch 62/600
1984/1984 [==============================] - 9s 5ms/step - loss: 0.1326 -
val_loss: 0.0694
Epoch 62: early stopping
```

[94]:
```
<keras.callbacks.History at 0x7fce885ebe20>
```

[95]:
```
losses = pd.DataFrame(model.history.history)
```

[96]:
```
losses.plot()
```

[96]:
```
<Axes: >
```



[97]:
```
probs = model.predict(X_test)
```

```
620/620 [==============================] - 1s 2ms/step
```

[98]:
```
preds = np.argmax(probs, axis=1)
```

[99]:
```
preds = preds + 1
```

[100]:
```
y_test
```

[100]:
```
array([3, 1, 1, …, 1, 2, 1])
```

[101]:
```
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           1       0.99      1.00      0.99     14688
           2       0.98      0.93      0.95      1358
           3       1.00      1.00      1.00       724
           4       0.98      0.99      0.98      1614
           5       0.99      0.99      0.99      1451

    accuracy                           0.99     19835
```

```
   macro avg       0.99      0.98      0.98     19835
weighted avg       0.99      0.99      0.99     19835
```

[102]:
```
print(confusion_matrix(y_test, preds))
```

```
[[14638    21     1    18    10]
 [   84  1257     1    11     5]
 [    1     0   722     1     0]
 [   10     9     0  1592     3]
 [   10     0     0     1  1440]]
```

[ ]:

[ ]:

[ ]:

[ ]: