Q3&Q4&Q5

November 29, 2021

0.0.1 Import libraries

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

0.0.2 Load the data

[ ]:
```
dataset = pd.read_csv('Data1.csv')
```

0.0.3 Split the data into train and test sets

[ ]:
```
# Split the dataset into Training and Test groups (use 20-80 split)
from sklearn.model_selection import train_test_split
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 0)
```

0.0.4 Feature scaling

[ ]:
```
# feature scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

0.0.5 Prepare reusable functions

[ ]:
```
def predict(model):
    y_pred = model.predict(X_test)
    # print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.
 ↪reshape(len(y_test),1)),1))
    return y_pred
```

[ ]:
```
def metrics(y_test, y_pred):
    from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report
```

```
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print("Accuracy score = {}".format(accuracy_score(y_test, y_pred)))

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

[ ]:
```
from sklearn.model_selection import cross_val_score
def k_fold(model, X, y, k):
    accuracies = cross_val_score(estimator=model, X=X_train, y=y_train, cv=k)
    print("accuracies = {}".format(accuracies))
    print("max accuracy = {}".format(accuracies.max()))
    print("mean = {}".format(accuracies.mean() * 100))
    print("std = {}".format(accuracies.std() * 100))
```

0.0.6 Logistic Regression

[ ]:
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lreg_classifier = LogisticRegression(random_state = 0)
lreg_classifier.fit(X_train, y_train)
```

[ ]:
```
LogisticRegression(random_state=0)
```

[ ]:
```
lreg_predictions = predict(lreg_classifier)
```

[ ]:
```
metrics(y_test, lreg_predictions)
```

```
[[84  3]
 [ 3 47]]
Accuracy score = 0.9562043795620438
              precision    recall  f1-score   support

           2       0.97      0.97      0.97        87
           4       0.94      0.94      0.94        50

    accuracy                           0.96       137
   macro avg       0.95      0.95      0.95       137
weighted avg       0.96      0.96      0.96       137
```

- There are 84 cases when the class was the class was 2 and is was correctly predicted
- There are 3 cases when the class was 2 and is was incorrectly predicted
- There are 3 cases when the class was 4 and it was incorrectly predicted
- There are 47 cases when the class was 4 and is was correctly predicted

The accuracy is almost 0.96. I may be overfitting.

[ ]:
```
k_fold(lreg_classifier, X_train, y_train, 10)
```

```
accuracies = [0.94545455 0.96363636 0.96363636 1.         0.94545455 1.
 0.96296296 0.96296296 0.98148148 0.94444444]
max accuracy = 1.0
mean = 96.70033670033669
std = 1.9697976894447813
```

The standard variation is low meaning that we can consider the mean accuracy as a good one. But the accuracies are highin some cases they are even 100%. This adaset is too small.

0.0.7 Decision Tree Classifier

[ ]:
```
# Decision tree classifier
from sklearn.tree import DecisionTreeClassifier
dtree_classifier = DecisionTreeClassifier(criterion = 'entropy', random_state =␣
 ↪0)
dtree_classifier.fit(X_train, y_train)
```

[ ]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

[ ]:
```
dtree_perdictions = predict(dtree_classifier)
```

[ ]:
```
metrics(y_test, dtree_perdictions)
```

```
[[85  2]
 [ 2 48]]
Accuracy score = 0.9708029197080292
              precision    recall  f1-score   support

           2       0.98      0.98      0.98        87
           4       0.96      0.96      0.96        50

    accuracy                           0.97       137
   macro avg       0.97      0.97      0.97       137
weighted avg       0.97      0.97      0.97       137
```

- There are 85 cases when the class was 2 and is was correctly predicted
- There are 2 cases when the class was 2 and is was incorrectly predicted
- There are 2 cases when the class was 4 and it was incorrectly predicted
- There are 48 cases when the class was 4 and it was correclty predicted

The accuracy score is bigger than the logistic regression accuracy. This accuracy is quite big.

[ ]:
```
k_fold(dtree_classifier, X_train, y_train, 10)
```

```
accuracies = [0.96363636 0.92727273 0.96363636 0.94545455 0.96363636 0.92727273
 0.90740741 0.96296296 1.         0.96296296]
max accuracy = 1.0
mean = 95.24242424242425
std = 2.4905835653388118
```

With desicion tree classifier we've got a lower mean accuracy but a higher standard deviation.

0.0.8 Random Forest Classifier

[ ]:
```
# Random Forest Classifier (with nb_trees = 10)
from sklearn.ensemble import RandomForestClassifier
rfc_classifier = RandomForestClassifier(n_estimators = 10, random_state = 0)
rfc_classifier.fit(X_train, y_train)
```

[ ]:
```
RandomForestClassifier(n_estimators=10, random_state=0)
```

[ ]:
```
rfc_predictions = predict(rfc_classifier)
```

[ ]:
```
metrics(y_test, rfc_predictions)
```

```
[[84  3]
 [ 3 47]]
Accuracy score = 0.9562043795620438
              precision    recall  f1-score   support

           2       0.97      0.97      0.97        87
           4       0.94      0.94      0.94        50

    accuracy                           0.96       137
   macro avg       0.95      0.95      0.95       137
weighted avg       0.96      0.96      0.96       137
```

Surprisengly the accuracy of the random forest classifier is same as the logistic regression accuracy and the consuption of the memory is also same as the logistic regression.

[ ]:
```
k_fold(rfc_classifier, X_train, y_train, 10)
```

```
accuracies = [0.94545455 0.96363636 0.94545455 1.         0.94545455 1.
 0.94444444 0.96296296 1.         0.94444444]
max accuracy = 1.0
mean = 96.51851851851852
std = 2.381554575703594
```

For k-fold cross validation we've got same results as in the case of logistic regression.

0.0.9 K-Nearest Neighbors (K-NN)

[ ]:
```
# K- Nearest Neighbors (K-NN)
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)
```

[ ]:
```
KNeighborsClassifier()
```

[ ]:
```
knn_predictions = predict(knn_classifier)
```

[ ]:
```
metrics(y_test, knn_predictions)
```

```
[[83  4]
 [ 2 48]]
Accuracy score = 0.9562043795620438
              precision    recall  f1-score   support

           2       0.98      0.95      0.97        87
           4       0.92      0.96      0.94        50

    accuracy                           0.96       137
   macro avg       0.95      0.96      0.95       137
weighted avg       0.96      0.96      0.96       137
```

- There are 83 cases when the class was 2 and it was correclty predicted
- There are 4 cases when the class was 2 and it was incorrectly predicted
- There are 2 cases when the class was 4 and it was incorrectly predicted
- There are 48 cases when the class was 4 and it was correctly predicted

The accuracy is almost the same as the accurancy given by the random forest classifier.

[ ]:
```
k_fold(knn_classifier, X_train, y_train, 10)
```

```
accuracies = [0.94545455 0.94545455 0.98181818 0.98181818 0.96363636 1.
 0.96296296 0.96296296 0.98148148 0.94444444]
max accuracy = 1.0
mean = 96.70033670033669
std = 1.7941421104663395
```

The accuracies are very simmilar to the ones obtained above. There is not much difference between all these classifiers in the way how they predict by using this dataset.

0.0.10 Naïve Bayes

[ ]:
```
# Naïve Bayes
from sklearn.naive_bayes import GaussianNB
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
```

[ ]:
```
GaussianNB()
```

[ ]:
```
naive_predictions = predict(nb_classifier)
```

[ ]:
```
metrics(y_test, naive_predictions)
```

```
[[80  7]
 [ 0 50]]
```

```
Accuracy score = 0.948905109489051
              precision    recall  f1-score   support

           2       1.00      0.92      0.96        87
           4       0.88      1.00      0.93        50

    accuracy                           0.95       137
   macro avg       0.94      0.96      0.95       137
weighted avg       0.96      0.95      0.95       137
```

For this model we've got a bit different result for the confusion matrix and a lower accuracy in comparison to the other classifiers.

- There are 80 cases when the class was 2 and it was correctly predicted
- There are 7 cases when the class was 2 and it was incorrectly predicted
- There are 0 cases when the class was 4 and it was incorrectly predicted
- There are 50 cases when the class was 4 and it was correctly predicted

[ ]:
```
k_fold(nb_classifier, X_train, y_train, 10)
```

```
accuracies = [0.92727273 0.96363636 0.96363636 0.96363636 0.96363636 0.96363636
 0.98148148 0.94444444 1.         0.94444444]
max accuracy = 1.0
mean = 96.15824915824916
std = 1.912472731957569
```

0.0.11 Support Vector Machine (SVM)

[ ]:
```
# Support Vector Machine (SVM)
from sklearn.svm import SVC
svm_classifier = SVC(kernel = 'linear', random_state = 0)
svm_classifier.fit(X_train, y_train)
```

[ ]:
```
SVC(kernel='linear', random_state=0)
```

[ ]:
```
svc_predictions = predict(svm_classifier)
```

[ ]:
```
metrics(y_test, svc_predictions)
```

```
[[83  4]
 [ 2 48]]
Accuracy score = 0.9562043795620438
              precision    recall  f1-score   support

           2       0.98      0.95      0.97        87
           4       0.92      0.96      0.94        50

    accuracy                           0.96       137
   macro avg       0.95      0.96      0.95       137
```

```
weighted avg       0.96      0.96      0.96       137
```

[ ]:
```
k_fold(svm_classifier, X_train, y_train, 10)
```

```
accuracies = [0.94545455 0.96363636 0.96363636 1.         0.94545455 1.
 0.98148148 0.96296296 1.         0.94444444]
max accuracy = 1.0
mean = 97.07070707070707
std = 2.1943977876398093
```

0.1 Q4

Choose the best performing model based on the results from performing the k-fold cross-validation. Discuss your choice.

Cross-validation results:

Logistic Regression: accuracies = [0.94545455 0.96363636 0.96363636 1. 0.94545455 1. 0.96296296 0.96296296 0.98148148 0.94444444]

max accuracy = 1.0

mean = 96.70033670033669

std = 1.9697976894447813

Decision tree classifier: accuracies = [0.96363636 0.92727273 0.96363636 0.94545455 0.96363636 0.92727273 0.90740741 0.96296296 1. 0.96296296]

max accuracy = 1.0

mean = 95.24242424242425

std = 2.4905835653388118

Random Forest Classifier (with nb_trees = 10): accuracies = [0.94545455 0.96363636 0.94545455 1. 0.94545455 1. 0.94444444 0.96296296 1. 0.94444444]

max accuracy = 1.0

mean = 96.51851851851852

std = 2.381554575703594

K-Nearest Neighbors (K-NN): accuracies = [0.94545455 0.94545455 0.98181818 0.98181818 0.96363636 1. 0.96296296 0.96296296 0.98148148 0.94444444]

max accuracy = 1.0

mean = 96.70033670033669

std = 1.7941421104663395

Naïve Bayes: accuracies = [0.92727273 0.96363636 0.96363636 0.96363636 0.96363636 0.96363636 0.98148148 0.94444444 1. 0.94444444]

max accuracy = 1.0

mean = 96.15824915824916

std = 1.912472731957569

Support Vector Machine (SVM): accuracies = [0.94545455 0.96363636 0.96363636 1. 0.94545455 1. 0.98148148 0.96296296 1. 0.94444444]

max accuracy = 1.0

mean = 97.07070707070707

std = 2.1943977876398093

To choose the best model I will take into account the mean accuracy and the standard deviation. The bigest mean accuracy is given by the SVM, but it has a bigger standard deviation than the other models. To choose the model, I will look for the gratest `mean accuracy - std`, because this would be the model that would have a bigger accuracy with lover standard deviation. If the mean accuracy is big andthe standarts devieation of the accurancies is small, then the model is good.

- Logistic regression: mean accuracy - std = 94.72
- Decision tree classifier: mean accuracy - std = 92.7518406771
- Random Forest Classifier: mean accuracy - std = 94.1369639428
- K-Nearest Neighbors (K-NN): mean accuracy - std = 94.9061945899
- Naïve Bayes: mean accuracy - std = 94.2457764263
- Support Vector Machine (SVM): mean accuracy - std = 94.8763092831

The best result is given by K-Nearest Neighbors (K-NN). This model gave us an accyracy of 100% just once in comparison with the other models, which is good because it is not desirable to have 100% accuracy for the model trained on the subsets.

0.2 Q5

KNeighborsClassifier hyperparameters chosen to tweak:

`n_neighbors` - we will tweak this one because a different number of neighbors might give us a different result, depending on how the data is distributed. This parameter represents how many neighbors will be taken in to accocunt when classifying a new data point.

`p` - we will tweak this one because it may change the results. Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

[ ]:
```
# List Hyperparameters that we want to tune
n_neighbors = [5, 10, 15, 20]
p=[1,2]
```

The only maccepted values for p are 1 and 2. I assume that p=2 (Euclidean distance) is the best choice because this distance is unique and it shows the shortest direct path.

I assume that if there are more neighbors taken into account, the model will be more accurate.

[ ]:
```
# GridSearch
from sklearn.model_selection import GridSearchCV

hyperparameters = dict(n_neighbors=n_neighbors, p=p)
clf = GridSearchCV(knn_classifier, hyperparameters, cv=10)
best_model = clf.fit(X_train, y_train)

print("Accucary score = {}".format(best_model.score(X_test, y_test)))
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.
 ↪get_params()['n_neighbors'])
```

```
Accucary score = 0.9562043795620438
Best p: 2
Best n_neighbors: 10
```

My assumption regarding p was correct. What about n_neighbours, I concluded that not alwasys more neighbors will make the model better.