MAD Assignment 3

Andreas V. W. Zacchi (nzl169)

December 14, 2024

Exercise 1

a)

Using PCA from the previous assignment we can plot the coordinates using the 2 dimensions with most variance. The coordinates can be seen in Figure 1

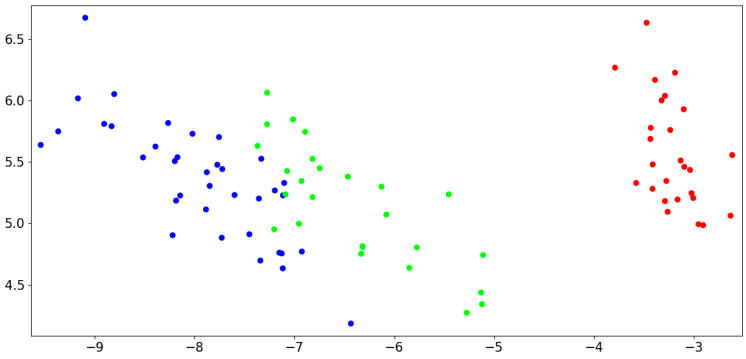


Figure 1: The two selected components by PCA which explains 97.67% of the variance

b)

After implementing kNN using the euclidean distance as metric and running it with (k=5) on the test data we obtain an accuracy of 100%, in the following plot the training data is plotted with the color corrensponding to it's type. A better plot including decision boundaries also with k=5 is shown later in subtask e).

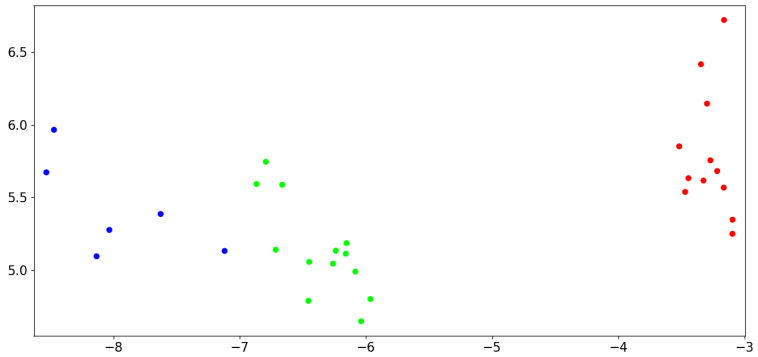


Figure 2: kNN with k=3 on the test data

c)

To select a good value of k, we try to run the function with some sensible k-values according to our data size which is only 30. The task says 1 to 5, and all n greater than 2 satisfy 100% accuracy, meaning those are the candidates.

We can remove 4 as we don't want to chose an even number as it may lead to ties. Both 3 and 5 are good choices, but I pick 5 as the labels are grouped pretty closely together with minor overlap of blue and green. Thus 5 instead of 3 might help in the overlapping region, however this is not certain.

d)

For this subtask I have, as recommended, used the `RandomForestClassifier`. By default it uses the entropy criterion and as our dataset is pretty small (30 samples) performance is not a concern. I also let it be at the default tree depth which splits until all leaves are pure. I also let it be at the default amount of features which is $\sqrt{D} = 2$, as $d = 1$ had worse accuracy for tree amount from 1 to 100.

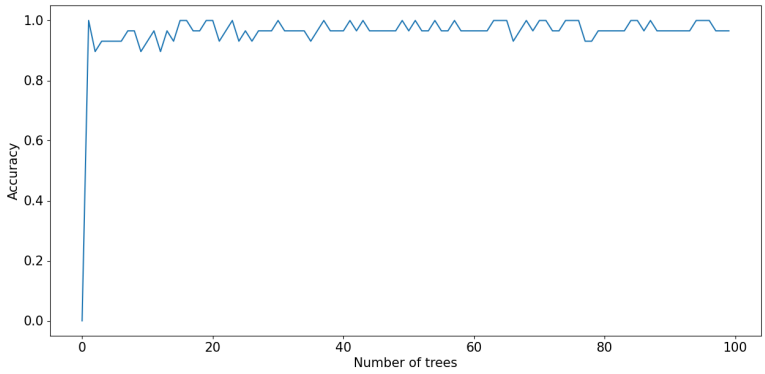


Figure 3: Plot plotting number of trees against accuracy

After running the code for plotting the above multiple times, as the trees are randomized a bit, I found that the best amount of trees was almost always above 10 and always greater than 14. Therefore I picked 15 as it has good accuracy (100% on validation) and decent performance.

e)

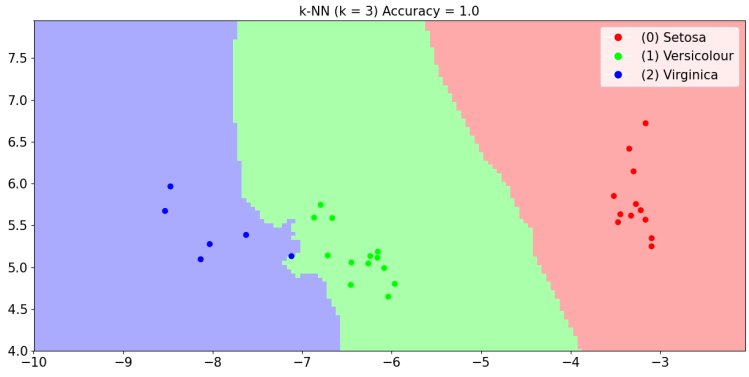


Figure 4: Plot showing the predictions for test data and decision boundaries for kNN

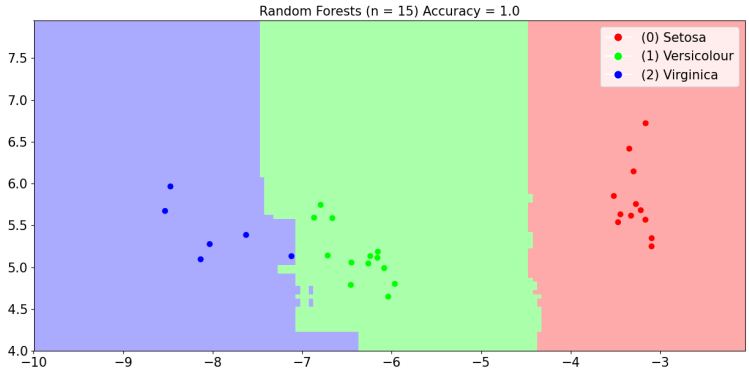


Figure 5: Plot showing the predictions for test data and decision boundaries for RF

Both kNN and the random forest has 100% accuracy on test data which suggests that they are good models for the data. Sometimes the random forest can have some blobs of green or blue inside the other side which is not ideal, and this problem isn't seen in kNN

Appendix

In [148…
```
# Load packages as usual
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import random
import matplotlib.cm as cm
import numpy.matlib
from matplotlib.colors import ListedColormap
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier


# Manipulating figure sizes
import matplotlib
matplotlib.rcParams['figure.figsize'] = (15,7)
matplotlib.rc('font', size=15)
matplotlib.rc('axes', titlesize=15)
```

In [149…
```
def __read(fileName, pTrainSamples = 0.6, pValidSamples = 0.2):
    emp_df = pd.read_csv(fileName)
    values = emp_df.values
    # Changed from np.float
    values = emp_df.values.astype(float)

    nTrainSamples = int(values.shape[0] * pTrainSamples)
    nValidSamples = int(values.shape[0] * pValidSamples)

    trainingFeatures = values[0:nTrainSamples, 0:-1]
    trainingLabels   = values[0:nTrainSamples, -1]
    validationFeatures = values[nTrainSamples:nTrainSamples + nValidSampl
    validationLabels   = values[nTrainSamples:nTrainSamples + nValidSampl
    testingFeatures    = values[nTrainSamples + nValidSamples:, 0:-1]
    testingLabels      = values[nTrainSamples + nValidSamples:, -1]
    # Changed from np.float and np.int
    return trainingFeatures.astype(float), trainingLabels.astype(int), \
           validationFeatures.astype(float), validationLabels.astype(int)
           testingFeatures.astype(float), testingLabels.astype(int)


trainingFeatures, trainingLabels, validationFeatures, validationLabels, t
print('shape training = ', trainingFeatures.shape)
print('shape validation = ', validationFeatures.shape)
print('shape testing = ', testingFeatures.shape)
```

```
shape training =  (88, 4)
shape validation =  (29, 4)
shape testing =  (31, 4)
```

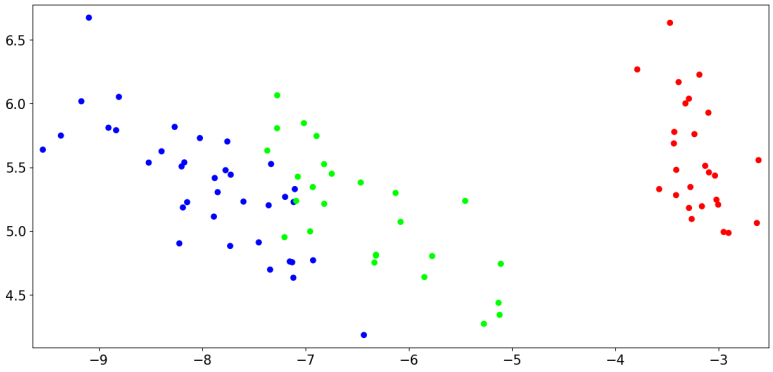
In [150…
```
def __PCA(data):
    data = data.T # Transpose the data so that each row is a feature (thi

    # Normalize the data
    mean = np.mean(data, 1) # Compute the mean of the data
    mean_cols = np.matlib.repmat(mean, data.shape[1], 1).T # Repeat the m
    data_cent = data - mean_cols # Center the data
```

```
    # Compute the covariance matrix
    covar = np.cov(data_cent)

    # Compute the eigenvectors and eigenvalues of the covariance matrix
    eigVals, eigVecs = np.linalg.eigh(covar) # Returned in ascending orde

    # Reverse the order of the eigenvalues and eigenvectors so that they
    PCevals = np.flip(eigVals) # Reverse the order of the eigenvalues
    PCevecs = np.flip(eigVecs, 1) # Reverse the order of the eigenvectors

    chosen_features = np.argmax(np.abs(PCevecs[:, :2]), axis=0)
    return PCevals, PCevecs


def __transformData(features, PCevecs):
    return np.dot(features,  PCevecs[:, 0:2])


PCevals, PCevecs = __PCA(trainingFeatures)
trainingFeatures2D = __transformData(trainingFeatures, PCevecs)
validationFeatures2D = __transformData(validationFeatures, PCevecs)
testingFeatures2D = __transformData(testingFeatures, PCevecs)
print('shape training = ', trainingFeatures2D.shape)
print('shape validation = ', validationFeatures2D.shape)
print('shape testing = ', testingFeatures2D.shape)
print('explained variance = ', np.sum(PCevals[0:2])/np.sum(PCevals))
```

```
shape training =  (88, 2)
shape validation =  (29, 2)
shape testing =  (31, 2)
explained variance =  0.9767350808869706
```

In [151…

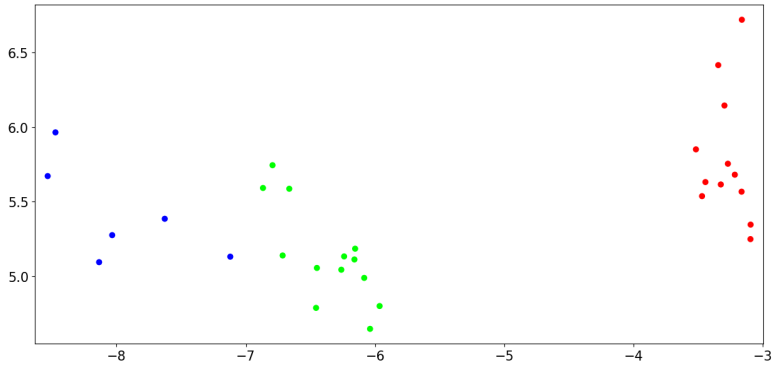
```
def __visualizeLabels(features, referenceLabels):
    plt.figure()
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold  = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
    y = referenceLabels

    plt.scatter(features[:, 0], features[:, 1], c = y, cmap = cmap_bold)
    plt.xlim(features[:, 0].min() - 0.1, features[:, 0].max() + 0.1)
    plt.ylim(features[:, 1].min() - 0.1, features[:, 1].max() + 0.1)
    plt.savefig('1a.png')
    plt.show()
    t = 0

__visualizeLabels(trainingFeatures2D, trainingLabels)
```



In [152…

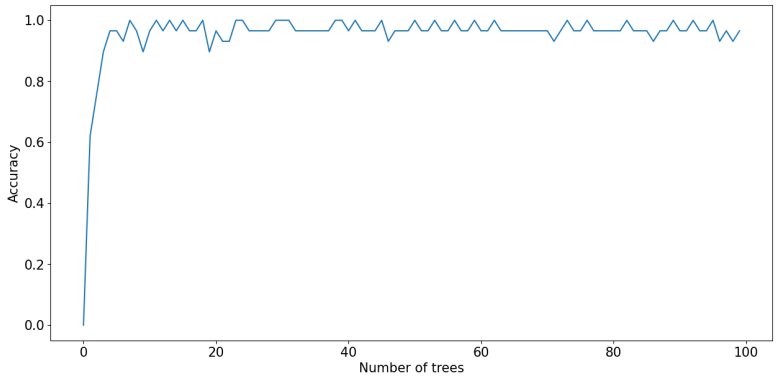
```python
def __kNNTest(trainingFeatures2D, trainingLabels, n_neighbors, validation
    predictions = []
    for validationFeature in validationFeatures2D:
        # Calcualte euclidean distances and sort them
        distances = np.sqrt(np.sum((trainingFeatures2D - validationFeatur
        sortedIndex = np.argsort(distances)

        # Take the n first labels
        closestLabels = trainingLabels[sortedIndex[0:n_neighbors]]

        # Get the most common label (bincount returns the number of occur
        prediction = np.argmax(np.bincount(closestLabels))
        predictions.append(prediction)

    if(plot):
        __visualizeLabels(validationFeatures2D, predictions)

    accuracy = np.sum(predictions == validationLabels) / len(validationLa
    return accuracy



# Best to pick 3 or 5 as both are odd numbers and have a good accuracy (1
for n in range(1, 6):
    print('accuracy (n = ', n, ') = ', __kNNTest(trainingFeatures2D, trai

__kNNTest(trainingFeatures2D, trainingLabels, 5, testingFeatures2D, testi
```

```
accuracy (n =  1 ) =  0.9310344827586207
accuracy (n =  2 ) =  0.9655172413793104
accuracy (n =  3 ) =  1.0
accuracy (n =  4 ) =  1.0
accuracy (n =  5 ) =  1.0
```



Out[152… 1.0

In [153…
```
def __randomForests(trainingFeatures2D, trainingLabels, n_trees = 100):
    # We can use entropy as the data set is not very large, we let the tr
    predictor = RandomForestClassifier(n_estimators=n_trees)
    predictor.fit(trainingFeatures2D, trainingLabels)
    return predictor


accuracy_list = np.array([0])
for n in range(1, 100):
    predictor = __randomForests(trainingFeatures2D, trainingLabels, n)
    predictions = predictor.predict(validationFeatures2D)
    accuracy = np.sum(predictions == validationLabels) / len(validationLa

    accuracy_list = np.append(accuracy_list, accuracy)

plt.figure()
plt.plot(accuracy_list)
plt.xlabel('Number of trees')
plt.ylabel('Accuracy')
plt.savefig('1d.png')
plt.show()

for n in range(1, 30):
    if(accuracy_list[n] == accuracy_list.max()):
        print('accuracy (n = ', n, ') = ', accuracy_list.max())
```

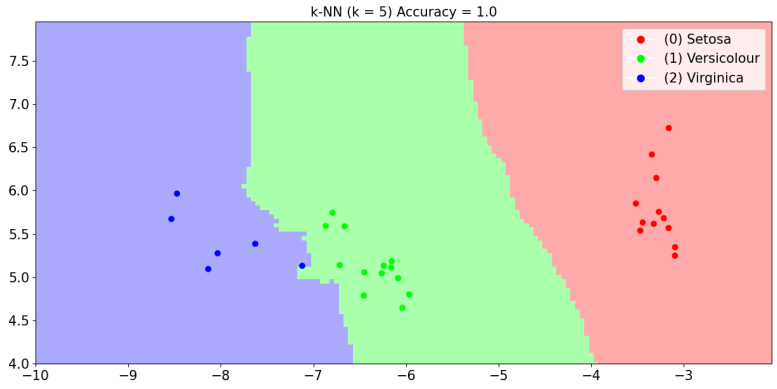


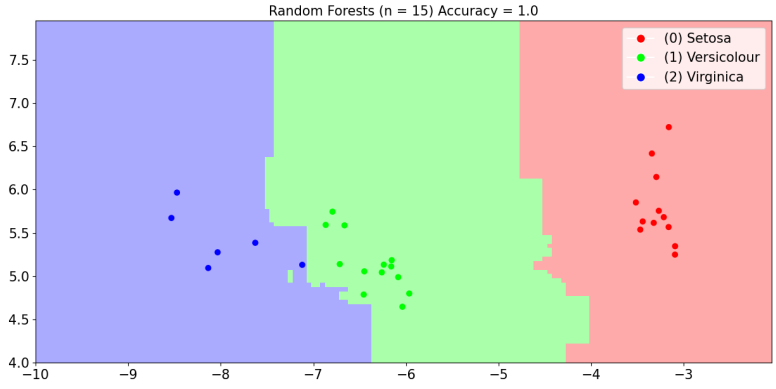

```
accuracy (n =  7 ) =  1.0
accuracy (n =  11 ) =  1.0
accuracy (n =  13 ) =  1.0
accuracy (n =  15 ) =  1.0
accuracy (n =  18 ) =  1.0
accuracy (n =  23 ) =  1.0
accuracy (n =  24 ) =  1.0
accuracy (n =  29 ) =  1.0
```

In [156…

```python
from matplotlib.lines import Line2D

def __kNN(trainingFeatures2D, trainingLabels, n_neighbors):
    predictor = KNeighborsClassifier(n_neighbors=n_neighbors)
    predictor.fit(trainingFeatures2D, trainingLabels)
    return predictor

def __visualizePredictions(predictor, features, referenceLabels):
    plt.figure()
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold  = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
    h = 0.05
    y = referenceLabels

    # x_min, x_max = features[:, 0].min() - 1, features[:, 0].max() + 1
    # y_min, y_max = features[:, 1].min() - 1, features[:, 1].max() + 1
    # xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
    #                      np.arange(y_min, y_max, h))
    # Z = predictor.predict(np.c_[xx.ravel(), yy.ravel()])
    # Z = Z.reshape(xx.shape)

    x0 = features[:, 0]
    x1 = features[:, 1]

    x0_min, x0_max = np.round(x0.min())-1, np.round(x0.max()+1)
    x1_min, x1_max = np.round(x1.min())-1, np.round(x1.max()+1)

    x0_axis_range = np.arange(x0_min,x0_max, h)
    x1_axis_range = np.arange(x1_min,x1_max, h)

    xx, yy = np.meshgrid(x0_axis_range, x1_axis_range)

    Z = predictor.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape) # Reshape the predictions to the meshgrid sha
```

```
    plt.pcolormesh(xx, yy, Z, cmap = cmap_light)
    # Plot also the training points
    plt.scatter(features[:, 0], features[:, 1], c = y, cmap = cmap_bold)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    # Add legend
    legend_elements = [Line2D([0], [0], marker='o', color='w', label='(0)
                       Line2D([0], [0], marker='o', color='w', label='(1)
                       Line2D([0], [0], marker='o', color='w', label='(2)
    plt.legend(handles=legend_elements, loc='upper right')

    # Add title depending on the predictor
    if isinstance(predictor, KNeighborsClassifier):
        plt.title('k-NN (k = ' + str(predictor.n_neighbors) + ')' + ' Acc
        plt.savefig('1e1.png')
    elif isinstance(predictor, RandomForestClassifier):
        plt.title('Random Forests (n = ' + str(predictor.n_estimators) +
        plt.savefig('1e2.png')

    plt.show()

k = 5
n_trees = 15
kNNPredictor = __kNN(trainingFeatures2D, trainingLabels, k)
RFPredictor  = __randomForests(trainingFeatures2D, trainingLabels, n_tree
__visualizePredictions(kNNPredictor, testingFeatures2D, testingLabels)
__visualizePredictions(RFPredictor, testingFeatures2D, testingLabels)
```




Random Forests (n = 15) Accuracy = 1.0
(0) Setosa
(1) Versicolour
(2) Virginica
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
−10
−9
−8
−7
−6
−5
−4
−3