Homework 2

May 1, 2019

1 DSE 220 Homework 2

1.0.1 Load libraries

```
In [1]: import numpy as np
        import pandas as pd
        import itertools
        import matplotlib.pyplot as plt
        from collections import Counter
        from sklearn import datasets
        from sklearn.model_selection import train_test_split
        from sklearn.mixture import GaussianMixture
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import confusion_matrix
        from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import mean_squared_error
        from sklearn.metrics import mean_absolute_error
        from sklearn.metrics import r2_score
```

1.1 Part 1 - Generative Learning

1.1.1 Load data

```
In [2]: # Load training dataset.
        X_train = pd.read_csv('./mnist_train_data.csv', delimiter=',',header=None)
        y_train = np.genfromtxt('./mnist_train_labels.csv', delimiter=',')

        # Load test dataset.
        X_test = pd.read_csv('./mnist_test_data.csv', delimiter=',',header=None)
        y_test = np.genfromtxt('./mnist_test_labels.csv', delimiter=',')
```

1.1.2 Question 1:

Compute and report the prior probabilities $_j$ for all labels. (10 marks)

```
In [3]: # Get labels
        labels=np.unique(y_train)
```

```
        # pi={label: prior probability...}
        pi={}
        for j in labels:
            pi[j] = np.sum(y_train==j)/len(y_train)
        pi
```

```
Out[3]: {0.0: 0.09866666666666667,
         1.0: 0.11183333333333334,
         2.0: 0.09683333333333333,
         3.0: 0.10133333333333333,
         4.0: 0.10383333333333333,
         5.0: 0.08566666666666667,
         6.0: 0.10133333333333333,
         7.0: 0.1085,
         8.0: 0.09183333333333334,
         9.0: 0.10016666666666667}
```

1.1.3 Question 2:

For each pixel X_i and label j, compute $P_{ji} = P(X_i = 1|y = j)$ (Use the maximum likelihood estimate shown in class). Use Laplacian Smoothing for computing Pji. Report the highest Pji for each label j. (15 marks)

```
In [4]: # Append 'class' label to traning dataset.
        X_train['class'] = y_train
```

Use the maximum likelihood estimation. $\hat{P}_{ji} = \frac{n_{ji}}{n_j}$

Laplacian Smoothing for P_{ji}: $\hat{P}_{ji} = \frac{n_{ji}+1}{n_j+2}$

```
In [5]: # Number of instaces of class j.
        n_j = X_train.groupby(['class']).size()

        # Number of instances of class j with Xi=1.
        n_ji = X_train.groupby(['class']).sum()

        # Get likelihood probabilitis.
        pr = (n_ji+1).div(n_j+2,axis=0)
```

```
In [6]: # Higest P_ji for each label j.
        pr.max(axis=1)
```

```
Out[6]: class
        0.0    0.851852
        1.0    0.985141
        2.0    0.728988
        3.0    0.808197
        4.0    0.849600
```

```
5.0    0.711240
6.0    0.849180
7.0    0.794793
8.0    0.875226
9.0    0.867330
dtype: float64
```

1.1.4 Question 3:

Use naive bayes (as shown in lecture slides) to classify the test data. Report the accuracy. (5 marks)

In [7]:
```
# Take the log of prior probabilities.
log_pi={}
for i in range(10):
    log_pi[i]=np.log(pi[i])
```

In [8]:
```
def NB(data):

    # pr_dictionary: {class j:{feature i: probability}...}
    pr_dict = pr.T.to_dict()

    # data_dictionary: {class j:{feature i: data}...}
    data_dict = data.T.to_dict()

    # Naive Bayes model.
    prediction = []
    for n in range(len(data)):
        score = {}
        for j in range(10):
            score[j] = 0
            for i in range(784):
                score[j] += np.log(pr_dict[j][i])*data_dict[n][i]
                score[j] += np.log(1-pr_dict[j][i])*(1-data_dict[n][i])
            score[j] += log_pi[j]

        max_score = max(score, key=score.get)
        prediction.append(max_score)

    return prediction
```

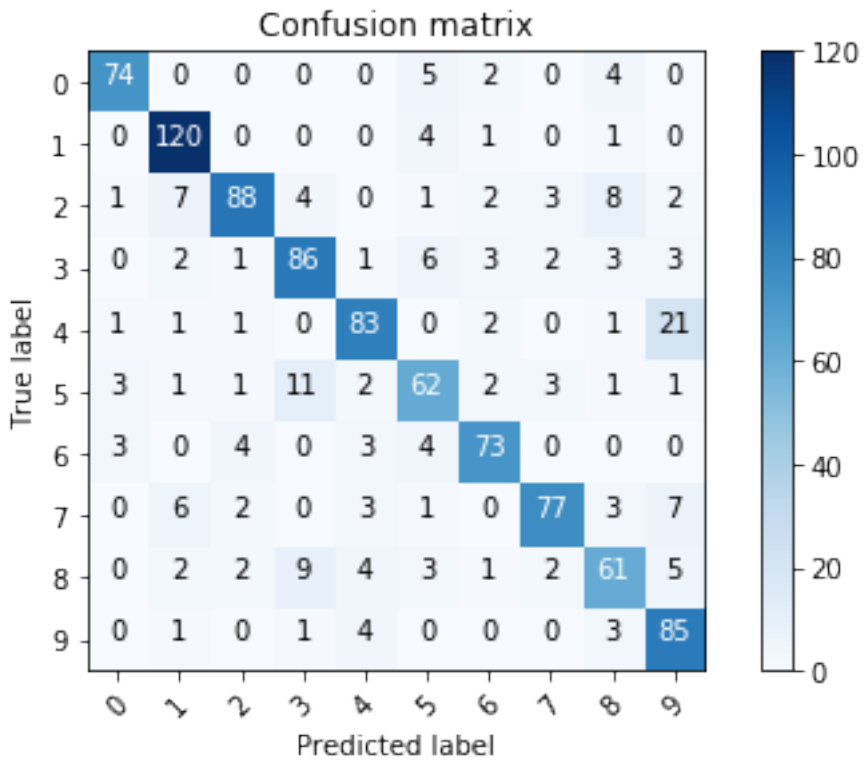
In [9]:
```
# Fit model on test dataset.
y_pred= NB(X_test)

# Test data accuracy.
accuracy = np.sum(y_pred==y_test)/len(y_test)
print('Accuracy of test data:',accuracy)
```

```
Accuracy of test data: 0.809
```

1.1.5 Question 4:

Compute the confusion matrix (as shown in the lectures) and report the top 3 pairs with most (absolute number) incorrect classifications. (10 marks)

```
In [10]: def plot_confusion_matrix(cm, classes,
                                   normalize=False,
                                   title='Confusion matrix',
                                   cmap=plt.cm.Blues):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """
             plt.imshow(cm, interpolation='nearest', cmap=cmap)
             plt.title(title)
             plt.colorbar()
             tick_marks = np.arange(len(classes))
             plt.xticks(tick_marks, classes, rotation=45)
             plt.yticks(tick_marks, classes)


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

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

In [11]: # Compute confusion matrix
         cnf_matrix = confusion_matrix(y_test, y_pred)
         np.set_printoptions(precision=2)

         # Plot non-normalized confusion matrix
         plt.figure()
         plot_confusion_matrix(cnf_matrix, classes=[*range(10)],title='Confusion matrix')
         plt.show()
```



In [12]:
```
# Convert confusion matrix to dictionary.
# incorrect{(true,prediction):count ...}
incorrect={}
for row in range(10):
    for col in range(10):
        if row!=col:
            incorrect[(row,col)]=cnf_matrix[row][col]

# Top 3 incorrect classification.
print("The top 3 incorrect classifications:\n")
top_3 = Counter(incorrect).most_common(3)
for item in top_3:
    print("Truth:",item[0][0],"Estimation:",item[0][1],"Count:",item[1])
```

```
The top 3 incorrect classifications:

Truth: 4 Estimation: 9 Count: 21
Truth: 5 Estimation: 3 Count: 11
Truth: 8 Estimation: 3 Count: 9
```

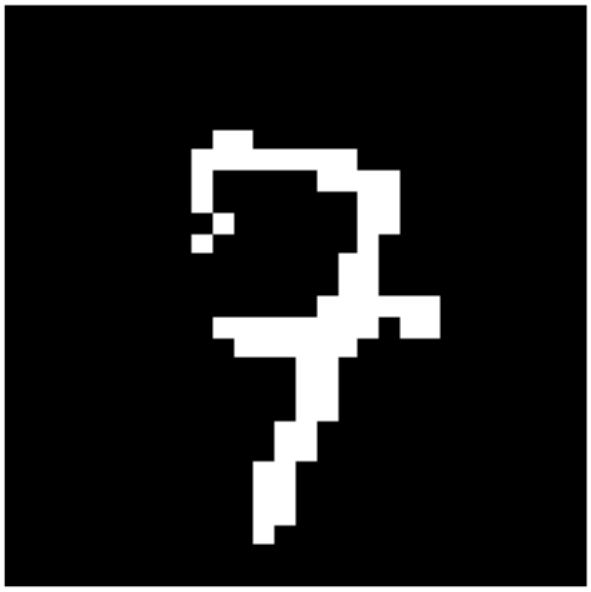
1.1.6 Question 5:

Visualizing mistakes: Print two MNIST images from the test data that your classifier misclassified. Write both the true and predicted labels for both of these misclassified digits. (10 marks)

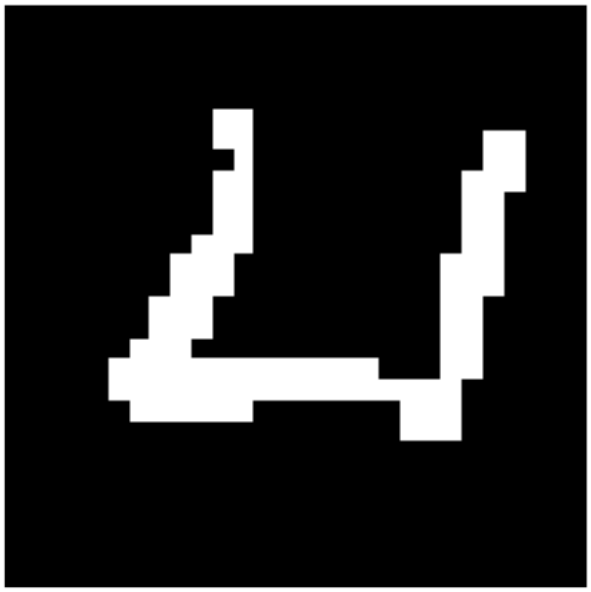
```
In [13]: def displaychar(image):
             plt.imshow(image.values.reshape([28,28]), cmap=plt.cm.gray)
             plt.axis('off')
             plt.show()
```

```
In [14]: # Random select two samples from the misclassifications.
         index = np.random.choice(list(*np.where(y_pred!=y_test)), 2)

         # Display the misclassified digit.
         for i in index:
             print("True label: {}; Predicted label: {};".format(y_test[i],y_pred[i]))
             displaychar(X_test.iloc[i,:])
```

```
True label: 7.0; Predicted label: 9;
```



```
True label: 4.0; Predicted label: 0;
```



1.1.7 Question 6:

Implement Gaussian Mixture model on the data as shown in class. Tune the covariance type parameter on the validation data. Use the selected value to compute the test accuracy. As always, train the model on train+validation data to compute the test accuracy. (10 mark)

In [15]:
```
# Load data from datasets.
data =datasets.load_breast_cancer()

X = data.data
y = data.target

# Split data in to tran-validation-test(40-20-40).
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.4, shuf
X_train, X_val, y_train, y_val= train_test_split(X_train_val, y_train_val, test_size=
```

In [16]:
```
# Model.
accuracy={}
covariance_type=['full', 'tied', 'diag', 'spherical']
for cov_type in covariance_type:
    clf = GaussianMixture(n_components=2, covariance_type=cov_type)

    clf.means_init = np.array([X_train[y_train == i].mean(axis=0) for i in range(2)])

    # Fit.
    clf.fit(X_train, y_train)
```

```
            # Predict.
            pred_val = clf.predict(X_val)

            # Evaluate.
            accuracy[cov_type]= accuracy_score(y_val, pred_val)
            print ('Covariance type: {},  validation accuracy = {}'.format(cov_type,(accuracy_
```

```
Covariance type: full,  validation accuracy = 0.6140350877192983
Covariance type: tied,  validation accuracy = 0.9035087719298246
Covariance type: diag,  validation accuracy = 0.9210526315789473
Covariance type: spherical,  validation accuracy = 0.9736842105263158
```

```
In [17]: # Select optimal covariance type.
         cov_type = max(accuracy,key=accuracy.get)

         # Model
         clf = GaussianMixture(n_components=2, covariance_type=cov_type)
         clf.means_init = np.array([X_train_val[y_train_val == i].mean(axis=0) for i in range(2
         # Fit
         clf.fit(X_train_val, y_train_val)

         # Predict.
         pred_test = clf.predict(X_test)

         # Evaluate.
         print ('Covariance type: {},  validation accuracy = {}'.format(cov_type,(accuracy_sco
```

```
Covariance type: spherical,  validation accuracy = 0.9385964912280702
```

1.1.8 Question 7:

Apply Linear Discriminant Analysis model on the train+validation data and report the accuracy obtained on test data. Report the transformation matrix (w) along with the intercept. (5 mark)

```
In [18]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

         # Intialize
         clf = LinearDiscriminantAnalysis()
         # Train
         clf.fit(X_train_val, y_train_val)
         # Test
         y_pred = clf.predict(X_test)

         # print the accuracy
         print ('Test accuracy = ' + str(np.sum(y_pred == y_test)/len(y_test)))
```

```
Test accuracy = 0.9736842105263158
```

```
In [19]: clf.coef_
```

```
Out[19]: array([[ 5.33e+00, -2.13e-01, -6.01e-01, -8.89e-03, -5.45e+01,  7.44e+01,
                 -4.98e+00, -3.42e+01,  1.73e+01, -4.45e+01, -8.41e+00, -2.53e-01,
                  5.79e-01,  1.06e-02, -2.81e+02, -2.63e+00,  4.26e+01, -1.67e+02,
                  1.27e+01,  2.01e+02, -3.54e+00, -7.22e-02, -5.54e-02,  2.39e-02,
                  1.39e+01,  1.04e+00, -6.06e+00, -1.48e+01, -1.63e+01, -8.05e+01]])
```

```
In [20]: clf.intercept_
```

```
Out[20]: array([50.96])
```

1.2 Part 2 - Evaluationg Classifier

1.2.1 Question 8:

Load sklearn's digits dataset (sklearn.datasets.load digits) and take the last 1300 samples as your test set. Train a K-Nearest Neighbor (k=5, linf distance) model and then without using any scikit-learn method, report the final values for Specificity, Sensitivity, TPR, TNR, FNR, FPR, Precision and Recall for Digit 3 (this digit is a positive, everything else is a negative). (15 marks)

Load data.

```
In [21]: # Load data from datasets.
         dataset =datasets.load_digits()

         X = dataset.data
         y = dataset.target

         # Split data into train-test set.
         X_train=X[:-1300]
         y_train=y[:-1300]

         X_test=X[-1300:]
         y_test=y[-1300:]
```

Fit KNN model.

```
In [22]: # Train k-nn model on train dataset.(k=5, l_inf distance(chebyshev))
         clf = KNeighborsClassifier(n_neighbors=5, metric='chebyshev')
         clf.fit(X_train, y_train)

         # Fit model on test dataset and get accuracy.
         y_pred = clf.predict(X_test)
         accuracy = np.sum(y_pred == y_test)/len(y_test)
         print('Test accuracy (k=5):',accuracy)
```

```
Test accuracy (k=5): 0.8938461538461538
```

Evaluations for digit 3.

```
In [23]: # Number of samples of digit 3 in test data.
         test_num = np.count_nonzero(y_test==3)
         pred_num = np.count_nonzero(y_pred==3)
         # print("Number of digit 3 in test data:",test_num)
         # print("Number of digit 3 in predicted data:",pred_num)


         # Indices of digit 3 in test and predicted labels.
         y_test_3 = np.where(y_test==3)
         y_pred_3 = np.where(y_pred==3)

         # TP, TN, FP, FN
         TP = np.sum(np.isin(y_test_3, y_pred_3))
         FN = np.sum(np.isin(y_test_3, y_pred_3,invert=True))
         FP = np.count_nonzero(y_pred==3)-TP
         TN = np.count_nonzero(y_pred!=3)-FN

         # Confusion matrix for digit 3.
         cm = pd.DataFrame(data = [[TP,FN],[FP,TN]],columns=['positive','negative'],index=['1'
         cm

Out[23]:    positive  negative
         1       113        17
         0        13      1157

In [24]: # Calculate evaluation matrics.
         specificity = TN/(TN+FP)
         sensitivity = TP/(TP+FN)
         TPR = TP/(TP+FN)
         TNR = TN/(TN+FP)
         FNR = FN/(FN+TP)
         FPR = FP/(FP+TN)
         precision = TP/(TP+FP)
         recall = TP/(TP+FN)
         print("Specificity={}\nSensitivity={}\nTPR={}\nTNR={}\nFNR={}\nFPR={}\nPrecision={}\nR
               .format(specificity,sensitivity,TPR,TNR,FNR,FPR,precision,recall))
```

```
Specificity=0.9888888888888889
Sensitivity=0.8692307692307693
TPR=0.8692307692307693
TNR=0.9888888888888889
FNR=0.13076923076923078
FPR=0.011111111111111112
Precision=0.8968253968253969
Recall=0.8692307692307693
```

1.3 Part 3 - Regression

An ablation experiment consists of removing one feature from an experiment, in order to assess the amount of additional information that feature provides above and beyond the others. For this section, we will use the diabetes dataset from scikit-learn's toy datasets. Split the data into training and testing data as a 90-10 split with random state of 10.

1.3.1 Question 9:

Perform least squares regression on this dataset. Report the mean squared error and the mean absolute error on the test data. (5 marks)

In [25]:
```
# Load data.
diabets =datasets.load_diabetes()

X = diabets.data
y = diabets.target

print(diabets.DESCR)
```

```
Diabetes dataset
================

Notes
-----

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

Data Set Characteristics:

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive values

  :Target: Column 11 is a quantitative measure of disease progression one year after baseline

  :Attributes:
    :Age:
    :Sex:
    :Body mass index:
    :Average blood pressure:
    :S1:
    :S2:
    :S3:
    :S4:
    :S5:
```

```
    :S6:

Note: Each of these 10 feature variables have been mean centered and scaled by the standard dev

Source URL:
http://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regressi
(http://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

```
In [26]: # Split data into train-test set.
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=

         # Least squares regression
         theta,residuals,rank,s = np.linalg.lstsq(X_train, y_train, rcond=None)

         # Make predictions on the test data
         prediction = np.dot(X_test, theta)
         mse = mean_squared_error(y_test, prediction)

         # Mean squared error.
         print ("Test data mean squared error:",mean_squared_error(y_test, prediction))

         # Mean absolute error.
         print ("Test data mean absolute error:",mean_absolute_error(y_test, prediction))
```

```
Test data mean squared error: 28060.622559310537
Test data mean absolute error: 160.8439534334583
```

1.3.2 Question 10:

Repeat the experiment from Question 9 for all possible values of ablation (i.e., removing the feature 1 only, then removing the feature 2 only, and so on). Report all MSEs. (10 marks)

```
In [27]: train={}
         test={}
         MSEs=[]
         for index in range(10):
             train[index] = np.delete(X_train,index,axis=1)
             test[index] = np.delete(X_test,index,axis=1)

             # Least squares regression
             theta,residuals,rank,s = np.linalg.lstsq(train[index], y_train, rcond=None)

             # Make predictions on the test data
```

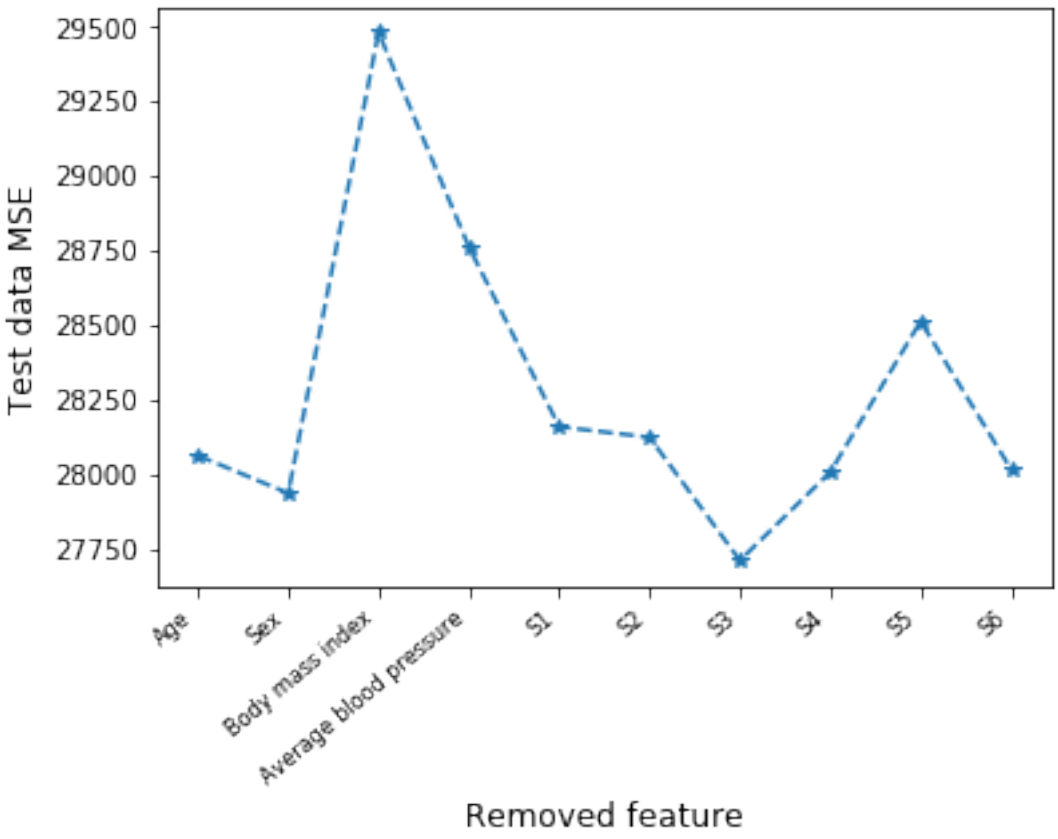
```
            prediction = np.dot(test[index], theta)
            MSE = mean_squared_error(y_test, prediction)
            MSEs.append(MSE)

            # Mean squared error.
            print ("Removed feature: {} , test data MSE: {}".format(index,MSE))
```

```
Removed feature: 0 , test data MSE: 28062.711822238078
Removed feature: 1 , test data MSE: 27936.317900620532
Removed feature: 2 , test data MSE: 29481.631561162365
Removed feature: 3 , test data MSE: 28762.573222191175
Removed feature: 4 , test data MSE: 28159.085211182188
Removed feature: 5 , test data MSE: 28123.775325632363
Removed feature: 6 , test data MSE: 27710.698098097317
Removed feature: 7 , test data MSE: 28006.79649746867
Removed feature: 8 , test data MSE: 28512.722055593065
Removed feature: 9 , test data MSE: 28017.248647620938
```

1.3.3 Question 11:

Based on the MSE values obtained from Question 10, which features do you deem the most/least significant and why? (5 marks)

```
In [28]: # Plot test data MSEs.
         features = ['Age','Sex','Body mass index','Average blood pressure','S1','S2','S3','S4
         plt.plot(features,MSEs,'*--')
         plt.xticks(rotation=40,ha='right',fontsize=8)
         plt.xlabel('Removed feature',fontsize=12)
         plt.ylabel('Test data MSE',fontsize=12)
         plt.show()
```



As we can see from the graph, the test data MSE increated sharply when we removed the feature of "Body mass index", and the MSE is the higest among all MSEs. Additionaly, the lowest MSE is removing the feature "S3". So we can assume that the feature `Body mass index` is **most** significant and the `S3` is **least** significant.