Student_Project_Example

April 20, 2021

1 CS-EJ3211 Machine Learning with Python

Student Project Example
Author: Andreas Fjellstad

1.1 Introduction

A microarray is a laboratory tool used to detect the expression of thousands of genes at the same time. DNA microarrays are microscope slides that are printed with thousands of tiny spots in defined positions, with each spot containing a known DNA sequence or gene.

The microarray data for this problem consists of normalized relative expression of certain genes measured in different tissue. There are 3000 gene probes and 2000 samples. The full dataset can be found at https://www.ebi.ac.uk/arrayexpress/ (accession number E-MTAB-62).

The first columns of 'data_subset.csv' file contains ID's of samples (e.g. 'GSM23227.CEL') and analyses info ('RMA') and the rest - expression values for 3000 genes.

The task is to predict the type of tissue ('disease' vs 'normal') based on expression profile of samples. In addition to this task, we solve ML problem of predicting multiple types of tissue {'cell line', 'disease', 'neoplasm', 'normal'}.

1.2 Problem formulation

The dataset consists of 2000 samples with 3000 expression values for genes in addition to information about the type of tissue, sample ID and analysis info. For this project each sample will be treated as a data point where the expression values will be the features and the tissue type the label. The project consists of two tasks, a main task and an extra task. In the main task, the aim is to develop a machine learning model that correctly predicts on the basis of its features whether a sample, that is, a data point, is labelled either 'disease' or 'normal'. In the extra task, the aim is to develop a machine learning model that correctly predicts on the basis of its features whether a data point is labelled either 'cell line','disease','neoplasm' or 'normal'.

The label space, i.e. the different type of tissues assigned to the samples, is thus in both the main task and in in the extra task discrete; in the main task the cardinality of the label space is 2 and in the extra task the cardinality of the label space is 4. This suggests that the problem should be understood as a classification problem.

As a classification problem, the project will utilise suitable metrics to evaluate the quality of the resulting machine learning models.

For the *main task* we will use F1 score for comparison and hypertuning of options, and then F1 score and confusion matrix in the final evaluation. With F1 score, the models will be compared with regard to the weighted average of precision and recall for the label designated as the positive label, where precision amounts the model's ability to avoid classifying a non-X sample as X while recall is the ability to classifying every X-sample as X. Together, this amounts a model's ability to avoid both overclassifying and underclassifying the label designated as positive. With 'disease' being translated into 1 and 1 being the default positive label, the F1 score evaluates a model's ability at neither overclassifying nor underclassifying samples labeled as 'disease'. In addition to F1 score, the final evaluation will also involve a normalised confusion matrix, that is, a $n \times n$ sized matrix where n is the number of labels in which each row displays the predicted distribution in percentage of labels for samples that should've been assigned that row's label.

For the *second task*, we will again use F1 score for the hypertuning of the model, however generalised to multiple labels in the sense that we compare the resulting models with regard to the F1 score for each label (as if we have n binary classification problems). In addition to F1 score, we will also use the balanced accuracy score for each option of parameters. While the accuracy score of a model is simply the fraction or number of correct predications, balanced accuracy score takes into account that the the dataset might be imbalanced, that is, that the distribution of labels is not even among the samples. For the final evaluation, we will employ again a confusion matrix in addition to the F1 score and the balanced accurancy score.

1.3 Methods

This section presents the methods used to solve the machine learning problem and walks through the process of solving the problem.

As mentioned above in the problem formulation, the dataset consists of 2000 samples with 3000 gene probes and a corresponding label representing the type of tissue, so that each sample will be represented by a data point consisting of 3000 features (the gene probes) and a label (type of tissue).

The kind of machine learning model we are after for the purposes of this task can be understood from a formal perspective as triple $(\mathcal{V}, \mathcal{L}, f)$ where $\mathcal{V}$ is a vector space, $\mathcal{L}$ is a label space and f a function from $\mathcal{V}$ to $\mathcal{L}$. The features of a data point are represented as a vector in $\mathcal{V}$ and the labels used to classify the data points are represented by elements of $\mathcal{L}$. The ideal aim is to find a function with the property that if it is applied to a vector in $\mathcal{V}$ which true label is in $\mathcal{L}$ correctly predicts that label. In practice, the aim is to find the function which offers the best trade-off between costs and correctness in the context where the machine learning model will be applied, where costs include not only computational costs in terms of algorithmic complexity and thus time, but also costs for example in terms of data storage, labour to gather and prepare data and also financial costs to pay for the required labour and equipment.

The first step is prepare the dataset. Since the sample ID and the procedure for analysing the sample is deemed irrelevant for the task at hand, and the main task concerns samples categorised as either 'normal' or 'disease', the first step after loading the file with the dataset is to remove the columns for the sample ID and the procedure, and moreover exclude every sample which is labelled either 'cell line' or 'neoplasm'. Once the data has been prepared in that way, the next step is to split the remaining samples into two non-empty subsets, a training set and a test set.

Both the training set and the test set will then consist of vectors from $\mathcal{V}$ with corresponding labels

from $\mathcal{L}$. The thought behind distinguishing between a training set and a test set is to employ the training set to find a proposal for a function, and then utilises the test set to check the extent to which the function found using the training set will provide the correct label when applied on vectors outside the training set. This would provide some evidence for that the model generalises to new data points. Indeed, depending on the kind of function one searches for and the method employed for finding that function, one could end up with a function that relies on aspects with the vectors that are not shared by other vectors sharing the same classification in the relevant vector space to the same extent, and that one thus ends up with a function that is *overfitted* to the training set. As seems to be usual based on the material for this course, the split will be 80/20 between the training set and the test set.

In the case where $\mathcal{V}$ consists of vectors of floating-point numbers, a number system for rational numbers used to represent of decimal numbers (and thus in practice real numbers) used in computer hardware, the set of possible functions is too big by itself to be searched through, and it is thus common to select a subset of that set and search through that subset for suitable functions.

For this task, we will search through two subsets, that explored with Binary Logistic Regression (LogReg) and a subset of that explored with Support Vector Machines (SVM) for classification based on radial basis function kernel. In the case of both LogReg and SVM, the basic assumption is that there is a linear relationship between the features and the basis for assigning the label.

In the case of Binary LogReg, it is assumed that the function is of the form

$$f(\mathbf{x}) = \begin{cases} 1 & w_0 + \mathbf{x} \cdot \mathbf{w} \geq 0 \\ 0 & w_0 + \mathbf{x} \cdot \mathbf{w} < 0 \end{cases}$$

where $\mathbf{x}$ is the vector representing the features of a data point, $\mathbf{w}$ a vector of weights $w_1, \ldots, w_n$ where n is the number of features in $\mathbf{x}$ and $\cdot$ is the dot product. This is justified because it is assumed that $w_0 + \mathbf{x} \cdot \mathbf{w} = \ln[p/(1-p)]$ where p is the probability that $\mathbf{x}$ is 1 and $\ln z$ the natural logarithm of z. The aim of the search becomes then to minimise the average logistic loss with respect to w_0 and $\mathbf{w}$ where the logistic loss of a data point with features $\mathbf{x}$ and label y is defined as

$$-y \ln[\sigma(w_0 + \mathbf{x} \cdot \mathbf{w})] - (1-y) \ln[1 - \sigma(w_0 + \mathbf{x} \cdot \mathbf{w})]$$

where $\sigma(z)$ is the sigmoid function $1/(1 + \exp(-z))$.

In the case of Support Vector Machines, a data point as a vector is treated explicitly as a point in an n-dimensional Euclidean space, and the aim is to find the hyperplane for that space which separates the points according to the class they belong to with the greatest margin which in turn will lead to a lower generalization error. This is achieved by selecting the hyperplane that the furthest away to the nearest point of any class. The points that are at or within distance to the hyperplane as given by greatest distance to the nearest point of any class become the support vectors that are utilised in the decision function for determining the class of a point. The goal of the "machine" then, is to find the vectors that support the classification. As it might not be the case that the classes are linearly separable within the n-dimension space, Support Vector Machines allow for the employment of a kernel function that effectively projects the data point into an implicit high- or infinite-dimensional space in which the classes will be linearly separable. For this task we will rely on a radial basis function (RBF) kernel which projects the data point into an implicit infinite-dimensional space.

The goal with a SVM then, is to find a $\mathbf{w}$ and w_0 such

$$f(\mathbf{x}) = \begin{cases} 1 & w_0 + \varphi(\mathbf{x}) \cdot \mathbf{w} \geq 0 \\ 0 & w_0 + \varphi(\mathbf{x}) \cdot \mathbf{w} < 0 \end{cases}$$

is correct for the most data points $\mathbf{x}$ among the training samples, where φ is a function such that $K(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{x}')$ where $K(\mathbf{x}, \mathbf{x}')$ is the kernel function which in our case is the radial basis function defined as $\exp(-\gamma|| \mathbf{x} - \mathbf{x}' ||^2)$ where $\gamma > 0$. This is achieved by minimising a variant of hinge loss involving a term C which then acts as a regularisation parameter. Given the access to an implicit infinite dimensional space, the SVM can deliver a perfect classification of any training set. However, a perfect classification of the training set might not be the best function for classifying the test set, and through C one permits some error in the classification of the training set. When fitting SVM on a training set then, it is important to try out different values for not only C but also γ. Due to the role C and γ have in finding the appropriate parameters of the decision function, they are referred to as hyperparameters.

In order to try out different settings for the hyperparameters, it is common to perform what is called a gridsearch. A gridsearch is basically a process in which one go systematically through a list of options for the hyperparameters in order to establish which parameters result in the best-performing model. When evaluating a setting for the hyperparameters, it is not the performance on a training set but rather the performance on a test set that matters. However, since one typically only has access to a limited training set, both the parameters found with some setting for the hyperparameters and the result of applying them on the test set will depend on exactly which samples are used for training and which are used for testing.

To overcome the latter issue, it is common to perform a k-fold cross validation for each proposed setting for the hyperparameters within the gridsearch. A k-fold cross validation amounts to first providing k many ways of splitting the training set into a new training/test pair, and then loop through each such pair by first training the model with the current setting on the training set of that pair and then testing it out on the test set of that pair. Each such model is evaluated and the scores are stored. After going through each split, the average score is calculated and then used to evaluate the current setting for the hyperparameters. After performing a k-fold for each proposed setting for the hyperparameters, the model that scores highest is selected.

A k-fold cross validation is also used with regard to binary logistic regression in order to obtain a score which can be compared with the score of the best model obtained with the grid search among the SVM models.

Importantly, the appropriate training set for each k-fold will not simply be fitted with either LogReg or SVM. Instead, the data will first be transformed in two steps. Firstly, each feature of the training set will be standardised using StandardScaler. StandardScaler calculates the mean and the standard deviation of each feature among the training samples, and then replaces the value of that feature of a sample with a score according to the formula $(x-u)/s$ where x is the original value, u the mean and s the standard deviation. The standardisation ensures that individual features are not prioritised or neglected in the search for a suitable function, as this ensures that the search concerns the variation among the features. When applying StandardScaler, it is important to distinguish between fitting a data set and transforming a data set. While fitting a data set amounts to establishing the relevant mean and standard deviation, transforming a data set amounts to applying the above formula to obtain new values. The features in the test set is only transformed through that formula using the u and s obtained with the training set, while the training set is first used to obtain the values and then transformed. After all, the test set is there for testing the model and cannot be used in obtaining the model. For the same reason it is important that each new training set in each of the k pair of a k-fold is separately fitted and transformed StandardScaler.

After the training set has been fitted and transformed using StandardScaler, we will also reduce its

dimensionality from 3000 to 20 using Principal Component Analysis (PCA). The dimensionality is reduced by establishing the dimensions along which the data displays the most variance and use these dimensions as features. The main purpose with this step is to reduce the computational load for the logistic regression and the SVM. As in the case of StandardScaler, at each pair in a k-fold the PCA will be fitted with the training set before the training set is transformed with the fitted PCA, while the test set will only be transformed with the PCA.

When the best-performing model has been selected between that based on binary logistic regression and the best among those explored through the grid search, the entire training set is used to fit and then transformed with StandardScaler and PCA before it is used to fit a model based then either on binary logistic regression or SVM with the best-performing hyperparameters. Finally then, we evaluate its performance using the test set that was split away from the training set before we engaged in the grid search and cross validation procedures. In that way, we can test the model obtained from fiting the training set on what can be described as 'previously unseen data' since the data in the test set was not used in the process of selecting the model.

[1]:
```
#importing classes
import numpy as np
import pandas as pd
from sklearn import␣
 ↪model_selection,preprocessing,decomposition,linear_model,pipeline,svm,metrics
import matplotlib.pyplot as plt
```

[2]:
```
#Loads the csv file, excludes columns [0,1] and features with labels 'cell␣
 ↪line' and 'neoplasm'
df = pd.read_csv('data_subset.csv')
df.drop(df.columns[[0,1]],axis=1,inplace=True)
df_trim = df.loc[np.logical_and(df['label'] != 'cell line',df['label'] !=␣
 ↪'neoplasm')]

#Splits the trimmed dataframe as converted to Numpy array into features and␣
 ↪labels
X,y=np.split(df_trim.to_numpy(),[df_trim.shape[1]-1],axis=1)
#redefine y from text to integer such that y_i=1 iff y_i='disease', and also␣
 ↪reshapes it into a vector
y=((y=='disease').astype(int)).reshape((-1,))
#splits the data into train and test sets
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,␣
 ↪test_size=0.2, random_state=42)
```

[3]:
```
#This cell defines a pipeline for LogReg according to the instructions and␣
 ↪reports the results

scaler = preprocessing.StandardScaler()
pca = decomposition.PCA(n_components = 20, random_state=42)

log_reg = linear_model.LogisticRegression()
```

```
pipe_lr = pipeline.Pipeline([('scaler', scaler), ('pca',pca),␣
 ↪('log_reg',log_reg)])
#cross_val_score returns a list, and the average is thus obtained as the mean␣
 ↪of that list:
log_reg_average_score = model_selection.
 ↪cross_val_score(pipe_lr,X_train,y_train,scoring='f1',cv=5).mean()
print(f"mean f1 score in LogReg model based on cross validation:␣
 ↪{round(log_reg_average_score,2)}")
```

```
mean f1 score in LogReg model based on cross validation: 0.62
```

[4]:
```
#This cell defines a pipeline for SVC according to the instructions and reports␣
 ↪the results

pipe_svr = pipeline.Pipeline([('scaler', scaler), ('pca',pca), ('svc', svm.
 ↪SVC())])
C_list=[0.01, 1, 100]
gamma_list=[1e-04, 1e-03, 1e-02]
params = {'svc__C': C_list,'svc__gamma': gamma_list}
cv = model_selection.
 ↪GridSearchCV(estimator=pipe_svr,param_grid=params,scoring='f1',␣
 ↪cv=5,return_train_score=True)
cv.fit(X_train,y_train)
svc_average_score = cv.best_score_

print(f"mean f1 score in SVC model with best params based on cross validation:␣
 ↪{round(svc_average_score,2)}")
```

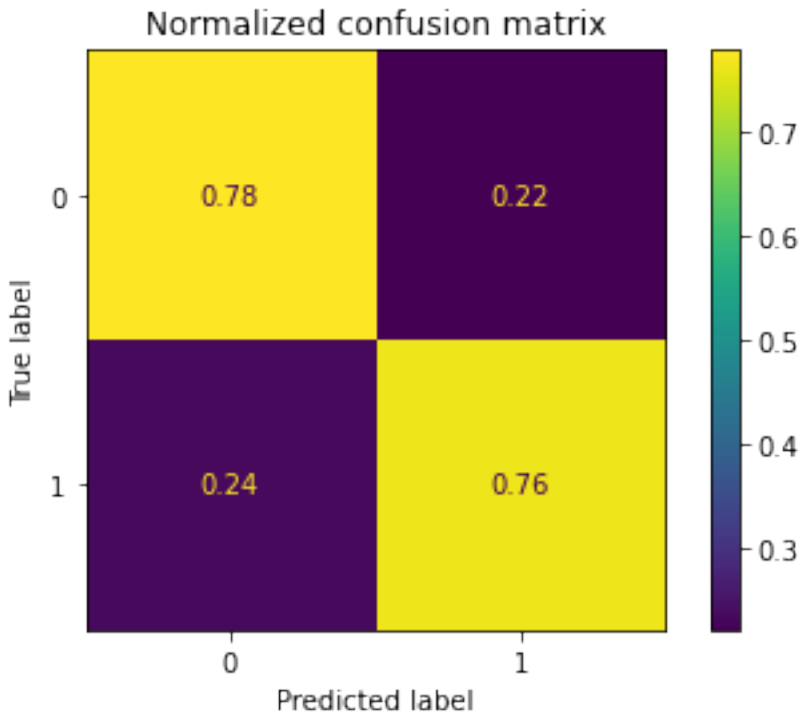
```
mean f1 score in SVC model with best params based on cross validation: 0.7
```

[5]:
```
#The following code selects the model with the highest mean f1 score:
print("best model:")
best_model = ('',None)
if log_reg_average_score > svc_average_score:
    best_model = ('log_reg',log_reg)
    print("LogisticRegression")
else:
    best_model = ('svc',svm.SVC(C=cv.best_params_['svc__C'],gamma=cv.
 ↪best_params_['svc__gamma']))
    print("SVC with C = ",cv.best_params_['svc__C']," and gamma = ",cv.
 ↪best_params_['svc__gamma'],sep='')

#A new pipeline is defined with the best model:
best_pipe = pipeline.Pipeline([('scaler', scaler), ('pca',pca), best_model])
best_pipe.fit(X_train,y_train)
y_pred_train = best_pipe.predict(X_train)
y_pred_test = best_pipe.predict(X_test)
```

```
#f1 and accuracy score from training and test with the new pipeline is␣
 ↪displayed:
print(f"f1 score on training set: {round(metrics.
 ↪f1_score(y_train,y_pred_train),2)}")
print(f"f1 score on test set: {round(metrics.f1_score(y_test,y_pred_test),2)}")
print(f"accuracy score on training set: {round(metrics.
 ↪accuracy_score(y_train,y_pred_train),2)}")
print(f"accuracy score on test set: {round(metrics.
 ↪accuracy_score(y_test,y_pred_test),2)}")
```

```
best model:
SVC with C = 100 and gamma = 0.001
f1 score on training set: 0.99
f1 score on test set: 0.76
accuracy score on training set: 0.99
accuracy score on test set: 0.77
```

[6]:
```
disp = metrics.plot_confusion_matrix(best_pipe, X_test, y_test,normalize='true')
disp.ax_.set_title("Normalized confusion matrix")
plt.show()
```



In this experiment we will perform a manual Grid Search with an embedded KFold for a multiclass scenario involving all four labels involving the same options for the hyperparameters are those explored in the binary scenario with Support Vector Machines for classification.

The models will be compared using three metrics. In addition to mean balanced accuracy score of the three KFolds for each option, we will also compare them with regard to the mean F1 score and the greatest least F1 score. The scores are obtained as follows. After each fold in a KFold the balanced accuracy score and the separate F1 scores for each class are collected, and after the three folds the average balanced accuracy score, the average separate F1 score for each class and the average F1 score across the four classes are calculated. We then use these metrics to select the model with the greatest least F1 score for some class, the model with the best mean F1 score and the best balanced accuracy score. As it turned out, each of the metrics favoured the same model.

The thought with including the greatest least separate F1 score was to have a quick test for whether there could be a model with similar F1 score average and balanced accuracy score, but perhaps more even distribution of separate F1 scores. A better metrics for that purpose could perhaps have been to measure the variance in addition to the mean of the separate F1 scores. However, since the quick test revealed the same result as the other two metrics I didn't see any point in programming a more complicated metrics.

[7]:
```
#Splits the DataFrame as converted to Numpy into features and labels
X,y =np.split(df.to_numpy(),[df.shape[1]-1],axis=1)
#redefine y from text to integer such that y_i=1 iff y_i='disease', and also␣
↪reshapes it into a vector
labels = {'normal': 0, 'disease': 1, 'neoplasm': 2, 'cell line': 3}
y = np.array(list(map(labels.get, y.flatten())))
#splits into training and test sets
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,␣
↪test_size=0.2, random_state=42)
```

[8]:
```
#kfold, scaler and pca objects defined
skf = model_selection.StratifiedKFold(n_splits=3,shuffle=True,random_state=42)
scaler = preprocessing.StandardScaler()
pca = decomposition.PCA(n_components = 20, random_state=42)

#variables for keeping track of best model
least_F1_result= 0
least_F1_params = (0.0,0.0)
mean_F1_result= 0
mean_F1_params = (0.0,0.0)
bal_acc_result=0.0
bal_acc_params=(0.0,0.0)

#manual gridsearch
for C in C_list:
    for gamma in gamma_list:
        svc = svm.SVC(C=C,gamma=gamma,class_weight='balanced')
        current_F1s=np.zeros((4,))
        current_bal_acc=0
        #the Kfold:
        for train_i, test_i in skf.split(X_train, y_train):
```

```
            svc.fit(pca.fit_transform(scaler.
↪fit_transform(X_train[train_i])),y_train[train_i])
            y_pred = svc.predict(pca.transform(scaler.
↪transform(X_train[test_i])))
            current_F1s +=metrics.f1_score(y_train[test_i],y_pred,average=None)
            current_bal_acc+=metrics.
↪balanced_accuracy_score(y_train[test_i],y_pred)
        #Averaging F1s and balanced accuracy
        current_F1s = current_F1s/3
        current_bal_acc = current_bal_acc/3
        print("rounded average F1 scores for each class with params␣
↪",(C,gamma),": ", [round(x, 2) for x in current_F1s])
        print("rounded balanced accuracy score with params ",(C,gamma),":␣
↪",round(current_bal_acc,2))
        #Checking if this model is better than any previously defined model:
        if min(current_F1s)>least_F1_result:
            least_F1_result=min(current_F1s)
            least_F1_params = (C,gamma)
        if sum(current_F1s)/4>mean_F1_result:
            mean_F1_result = sum(current_F1s)/4
            mean_F1_params = (C,gamma)
        if current_bal_acc>bal_acc_result:
            bal_acc_result=current_bal_acc
            bal_acc_params = (C,gamma)

#Printing results:
print("")
print("results:")
print("greatest least F1 score for some params:", round(least_F1_result,2))
print("obtained with params C =", least_F1_params[0], " and gamma␣
↪=",least_F1_params[1])
print("")
print("best mean F1 score with some params:",round(mean_F1_result,2))
print("obtained with params C =", mean_F1_params[0], " and gamma␣
↪=",mean_F1_params[1])
print("")

print("best balanced accuracy score:", round(bal_acc_result,2))
print("obtained with params C =", bal_acc_params[0], " and gamma␣
↪=",bal_acc_params[1])
```

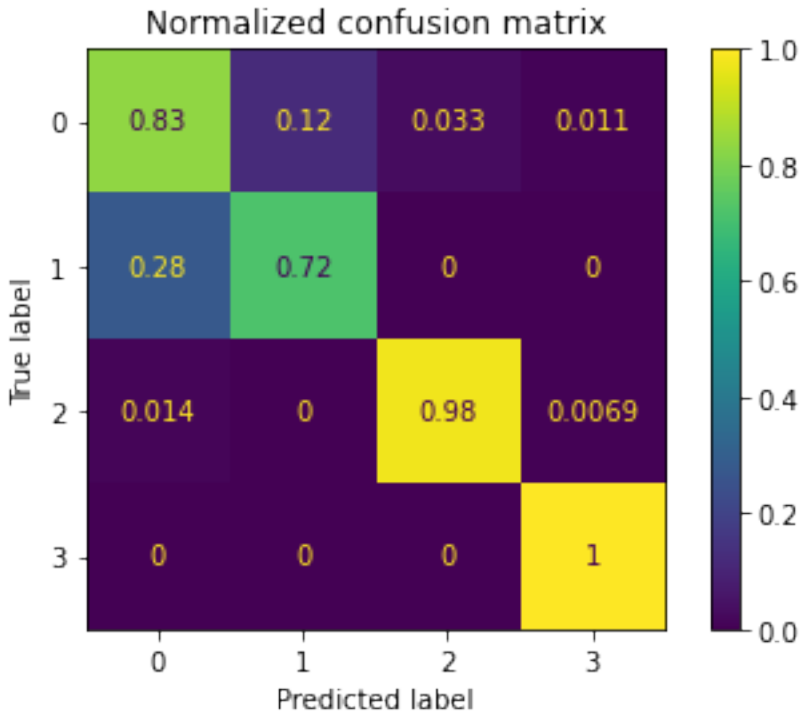
```
rounded average F1 scores for each class with params  (0.01, 0.0001) :  [0.0,
0.56, 0.0, 0.51]
rounded balanced accuracy score with params  (0.01, 0.0001) :  0.45
rounded average F1 scores for each class with params  (0.01, 0.001) :  [0.0,
0.58, 0.5, 0.53]
rounded balanced accuracy score with params  (0.01, 0.001) :  0.49
```

```
rounded average F1 scores for each class with params  (0.01, 0.01) :  [0.0, 0.0,
0.0, 0.4]
rounded balanced accuracy score with params  (0.01, 0.01) :  0.25
rounded average F1 scores for each class with params  (1, 0.0001) :  [0.48,
0.64, 0.94, 0.95]
rounded balanced accuracy score with params  (1, 0.0001) :  0.77
rounded average F1 scores for each class with params  (1, 0.001) :  [0.65, 0.69,
0.98, 0.98]
rounded balanced accuracy score with params  (1, 0.001) :  0.83
rounded average F1 scores for each class with params  (1, 0.01) :  [0.66, 0.58,
0.85, 0.86]
rounded balanced accuracy score with params  (1, 0.01) :  0.71
rounded average F1 scores for each class with params  (100, 0.0001) :  [0.7,
0.69, 0.98, 0.99]
rounded balanced accuracy score with params  (100, 0.0001) :  0.85
rounded average F1 scores for each class with params  (100, 0.001) :  [0.76,
0.71, 0.98, 0.99]
rounded balanced accuracy score with params  (100, 0.001) :  0.86
rounded average F1 scores for each class with params  (100, 0.01) :  [0.69,
0.57, 0.85, 0.87]
rounded balanced accuracy score with params  (100, 0.01) :  0.72

results:
greatest least F1 score for some params: 0.71
obtained with params C = 100  and gamma = 0.001

best mean F1 score with some params: 0.86
obtained with params C = 100  and gamma = 0.001

best balanced accuracy score: 0.86
obtained with params C = 100  and gamma = 0.001
```

[9]:
```python
#Defining SVC object:
svc = svm.SVC(C=mean_F1_params[0],gamma=mean_F1_params[1])

#fitting training set to transformers and transforming training and test set
X_train_trans =  pca.fit_transform(scaler.fit_transform(X_train))
X_test_trans = pca.transform(scaler.transform(X_test))

#fitting training set
svc.fit(X_train_trans,y_train)

#presenting results:
train_f1 = metrics.f1_score(y_train,svc.predict(X_train_trans),average=None)
print("f1 scores on training set: ", [round(x, 2) for x in train_f1])
print("mean f1 score on training set: ", round(train_f1.mean(),2))
```

```
y_test_pred = svc.predict(X_test_trans)
test_f1 = metrics.f1_score(y_test,y_test_pred,average=None)
print("f1 scores on test set: ", [round(x, 2) for x in test_f1])
print("mean f1 score on test set: ", round(test_f1.mean(),2))
print("Balanced accuracy score on test set: ",round(metrics.
 ↪balanced_accuracy_score(y_test,y_test_pred),2))

disp = metrics.plot_confusion_matrix(svc,X_test_trans, y_test,normalize='true')
disp.ax_.set_title("Normalized confusion matrix")
plt.show()
```

```
f1 scores on training set:  [0.98, 0.98, 1.0, 1.0]
mean f1 score on training set:  0.99
f1 scores on test set:  [0.82, 0.76, 0.98, 0.99]
mean f1 score on test set:  0.89
Balanced accuracy score on test set:  0.88
```



1.4 Results

In the first experiment involving the binary scenario we observed that a model obtained through Support Vector Machines with hyperparameters C=100 and gamma=0.001 perform better with regard to F1 score than a model obtained logistic regression in the binary scenario in a comparison based on a 5-fold cross validation involving the training set (0.7 vs 0.62). On the test set, the former model achieved an F1 score of 0.76 and an accuracy score of 0.77. As the normalised confusion

matrix shows, the model correctly classifies with regard to the test set 78% of samples labelled 'normal' and 76% of samples labelled 'disease'.

In the second experiment involving the multilabel scenario we observed that Support Vector Machines with the same hyperparameters as those found performing better in the first experiment also came out as better performing in the multiclass scenario involving all four labels and the complete data set. In particular, through a grid search with a 3-fold cross validation it was shown that the model with best balanced accuracy score, best mean F1 score across the four labels and greatest least F1 score on some label was the model obtained with hyperparameters C=100 and gamma=0.001. Moreover, we observe that the model performed quite similar on the test set with regard to the labels 'normal' and 'disease' as the model obtained in the binary scenario, but performed more or less flawlessly with regard to the remaining two labels on the training set. This can be seen both on the F1 score for each label and in the confusion matrix.

Indeed, while we could in both experiments observe not only the ability of Support Vector Machines with RBF kernel to overfit the training set, but also how the cross validation ensured that the model generalised decently to the test set despite the overfitting. In the binary scenario, F1 and accuracy dropped from .99 to .76 and .77. In the multilabel scenario, there is a similar drop in F1 score and accuracy could be observed with the 'normal' and 'disease' samples. However, in the case of 'cell line' and 'neoplasm', the generalisation from training to test set was seemingly perfect in the sense that the model correctly classified respectively 99% and 100% of those samples in the test set. This is an odd result and will be adressed in the discussion below.

1.5 Discussion/Conclusions

In the evaluation of the model, there are two questions that immediately present themselves.

Firstly, one might ask whether the performance in the binary scenario or also in the multiclass scenario on test samples labelled 'normal' and 'disease' could be improved on. For example, in the multiclass scenario the model only correctly classifies 72% of the 'disease'-labeled samples.

Secondly, one might ask whether there is something weird going on with the samples labelled 'cell line' and 'neoplasm'. After all, the model classifies correctly 99% and 100% of those samples in the test set respectively, and that seems rather too fantastic. While this is a result we could expect for the training set since the function is based on a Support Vector Machine with RBF kernel, it is rather surprising for the test set.

With regard to the first question, the SKLearn documention suggests two simple way to improve the model. Firstly, it is suggested that Support Vector Machines with RBF kernel would benefit from PCA with the additional parameter 'whiten=True'. However, briefly playing with that parameter within the above code has shown that this does indeed lead the changes with regard to what would count as the better model in the cross validation phase (and the three metrics no longer agree on what's the better model), the better model based on balanced accuracy in the cross validation phase actually perform worse in the final evaluation. This is thus a parameter that could be explored in more detail, and could thus be a potential venue of further research. Secondly, the distribution of labels among the samples is not even, and it could thus be the case that the performance is improved with the additional parameter "class_weight='balanced' " in SVC, as this would ensure that each label is weighted equally despite their ueven distribution among the samples. However, a quick check within the above code revealed that adding this parameter does not lead to any immediate improvement on the results.

In addition to these two rather straightforward ways of improving the model with regard to the classification of samples labeled 'disease' and 'normal', it could also be the case that better models are found by extending the scope of the grid search. While the SVM documentation in SKLearn advices to keep values in the gridsearch exponentially far apart, the search could clearly be expanded 'upwards'. However, some quick testing within the above code revealed that also here things do not immediately improve. For example, while one can obtain models that score better in the cross validation phase, these models do not score better in the final evaluation.

With the immediate ways of modifying the model not leading to improved performance, it seems reasonable to question whether the data set itself is faulty in some way. For example, it could be the case that samples labelled 'normal' do not have anything in particular in common, and similarly with 'disease'. In other words, while the samples labelled 'normal' and 'disease' might have sufficient 'family resemblance' in order to achieve the above-mentioned performance, they aren't sufficiently similar for that performance to be improvable without the addition of new features.

Moreover, the inverted scenario might also be what explains the 99% and 100% performance on the test set with samples labelled 'cell line' and 'neoplasm', namely that those sets of samples have some features in common which is captured by the Support Vector Machine. This could even be the problem with those samples in the sense that they are too similar, and from which it would follow that the model wouldn't generalise well to samples that are outside the data set.

Now, labels such as 'cell line' and 'neoplasm' seems to concern something about tissues that are more "natural" than 'normal' and 'disease', and we could thus expect it to be more likely that the former labels correspond to distinguishable features with the samples. The difference in performance between the labels could thus have a "natural" cause, so to speak. While 'normal' and 'disease' are normative evaluations of the tissue, 'cell line' and 'neoplasm' would seem to be descriptive evaluations of the tissue. Another relevant distinction could be that a classification according to 'normal' and 'disease' rely not only on features intrinsic to the tissue but also features that are extrinsic to them and thus not captured within the data set. A classification according to 'cell line' and 'neoplasm', on the other hand, rely solely on features that are intrinstic to the tissue and thus captured by the data set. Thus, while the former could certainly correspond to statistically relevant differences which in turn explains the performance, it is precisely because they do not correspond to a descriptive evaluation that there is little potential for improving the performance. With regard to 'cell line' and 'neoplasm', on the other hand, the extreme performance could be explained in terms of how the labels concern something intrisic with the tissue samples. It is even tempting to conclude that either the latter is the case or the samples provided in the data set are too similar.

In any case, the next logical step is therefore to dig deeper into the data set in order to attempt to clarify whether the data set itself is useful for developing a model for classifying samples according to those classifications. One way to shed some light on this issue is by exploring whether unsupervised methods for clustering the data points would deliver clusters that correspond more or less to the labels. While there are different methods available that would give different results, one reasonable hypothesis based on the above discussion is that at least some of the methods would deliver clusters that correspond more or less to 'cell line' and 'neoplasm', but not to 'normal' and 'disease'.