Assignment 5: Ensemble Methods

Copyright and Fair Use

Automatic Testing Guidelines

Automatic unittesting requires you, as a student, to submit a notebook which contains strictly defined objects. Strictness of definition consists of unified shapes, dtypes, variable names and more.

Within the notebook, we provide detailed instruction which you should follow in order to maximise your final grade.

Name your notebook properly, follow the pattern in template name:

Assignment_N_NameSurname_matrnumber

1. N - number of assignment
2. NameSurname - your full name where every part of the name starts with a capital letter, no spaces
3. matrnumber - your 8-digit student number on ID card (without k)

Example:
✅ Assignment_0_RenéDescartes_12345678
✅ Assignment_0_SørenAabyeKierkegaard_12345678
❌ Assignment0_Peter_Pan_k12345678

Don't add any cells but use the ones provided by us. You may notice that most cells are tagged such that the unittest routine can recognise them.

We highly recommend you to develop your code within the provided cells. You can implement helper functions where needed unless you put them in the same cell they are actually called. Always make sure that implemented functions have the correct output and given variables contain the correct data type. Don't import any other packages than listed in the cell with the "imports" tag.

Note: Never use variables you defined in another cell in your functions directly; always pass them to the function as a parameter. In the unitest they won't be available either.

Good luck!

Task 1: AdaBoostM1 is an instance of forward stagewise modelling

In the lecture it was mentioned that one of the first boosting algorithms, i.e. AdaBoostM1, is equivalent to forward stagewise modelling using the exponential loss $L(y, g(\mathbf{x})) = \exp(-yg(\mathbf{x}))$ for a binary classification problem. In this task we intend to provide proof of this fact. We will guide you through the most important steps and you will have to add some details.

For AdaBoostM1, the basis functions at timestep n are the individual classifiers $b_n(\mathbf{x}) \in \{-1, 1\}$. We assume that all of them are slightly better than random guessing. Note that we use b_n here for the resulting classifier at timestep n, which differs slightly from the notation in the slides, mainly to not confuse it with the corresponding approximation from forward stagewise modelling, which is also called g_n there.

Using the exponential loss in each timestep n we have to solve

$$(\beta_n, b_n) = \arg\min_{\beta,b} \sum_{i=1}^{l} \exp(-y_i(g_{n-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i)),$$

for the classifier b_n and the coefficient β_n which are added at each step. This can be rewritten as

$$(\beta_n, b_n) = \arg\min_{\beta,b} \sum_{i=1}^{l} w_i^{(n)} \exp(-y_i \beta b(\mathbf{x}_i)), \qquad (1)$$

with $w_i^{(n)} = \exp(-y_i g_{n-1}(\mathbf{x}_i))$. Since each $w_i^{(n)}$ depends neither on β nor b, it can be regarded as a weight that is applied to each observation. This weight depends on $g_{n-1}(\mathbf{x}_i)$, and so the individual weight values change with each iteration n. The solution of (1) can be found in two steps:

Calculation 1 (10 points):

In the **first step** fix $\beta \geq 0$ and show that in this case the solution to (1) is $b_n = \arg\min_b \sum_{i=1}^{l} w_i^{(n)} I(b(\mathbf{x}_i) \neq y_i)$. Hints:

1. Try to write the expressions in (1) after $\arg\min$ in the form $\sum_{b(\mathbf{x}_i)=y_i}(\dots) + \sum_{b(\mathbf{x}_i)\neq y_i}(\dots)$.
   Find the right expressions for (...), such that the product $b(\mathbf{x}_i)y_i$ doesn't appear there anymore.

2. Now show that this can be written as
   $(\exp(\beta) - \exp(-\beta)) \sum_{b(\mathbf{x}_i)\neq y_i} w_i^{(n)} + \exp(-\beta) \sum_{i=1}^{l} w_i^{(n)}$. (2)
   The equation $\sum_i w_i^{(n)} = \sum_{b(\mathbf{x}_i)\neq y_i} w_i^{(n)} + \sum_{b(\mathbf{x}_i)=y_i} w_i^{(n)}$ might be helpful.

3. Argue why this already implies the claim.

Please provide reasoning and explanations in full sentences. Grading of the task will heavily depend on it.

*Write your first step** calculation here*

1. Hint 1 ...
2. Hint 2 ...
3. Hint 3 ...

$$1.\ (\beta_n, b_n) = \arg\min_{\beta,b} \sum_{i=1}^{l} w_i^{(n)} \exp(-y_i \beta b(\mathbf{x}_i))$$
$$\text{Splitting between datapoints correctly classified } y_i b(\mathbf{x}_i) = 1 \text{ and } y_i b(\mathbf{x}_i) = -1$$
$$(\beta_n, b_n) = \sum_{b(\mathbf{x}_i)=y_i} w_i^{(n)} \exp(-\beta) + \sum_{b(\mathbf{x}_i)\neq y_i} w_i^{(n)} \exp(-\beta)$$

$$2.\ \text{Using} \sum_i w_i^{(n)} = \sum_{b(\mathbf{x}_i)\neq y_i} w_i^{(n)} + \sum_{b(\mathbf{x}_i)=y_i} w_i^{(n)}$$
$$\Rightarrow \sum_{b(\mathbf{x}_i)=y_i} w_i^{(n)} = \sum_i w_i^{(n)} - \sum_{b(\mathbf{x}_i)\neq y_i} w_i^{(n)}$$
$$(\beta_n, b_n) = \sum_{i=1}^{l} w_i^{(n)} \exp(-\beta) - \sum_{b(\mathbf{x}_i)\neq y_i} w_i^{(n)} \exp(-\beta) + \sum_{b(x_i)\neq y_i} w_i^{(n)} \exp(\beta)$$
$$= \sum_{i=1}^{l} w_i^{(n)} \exp(-\beta) + (\exp(\beta) - \exp(-\beta)) \sum_{b(\mathbf{x}_i)\neq y_i} w_i^{(n)}$$
$$= (\exp(\beta) - \exp(-\beta)) \sum_{b(\mathbf{x}_i)\neq y_i} w_i^{(n)} + \exp(-\beta) \sum_{i=1}^{l} w_i^{(n)}$$

$$3.\ \text{The expression from (1) can be rewritten to } b_n = \arg\min_b \sum_{i=1}^{l} w_i^{(n)} I(b(\mathbf{x}_i) \neq y_i)$$
$$\text{because it can be minimzed only if the sum of the weights is minimized, hence it makes sense to rewrite it with the identity matrix.}$$

Calculation 2 (20 points):

In the **second step** you need to optimize the following expression with respect to β:

$$(\exp(\beta) - \exp(-\beta)) \sum_{b_n(\mathbf{x}_i)\neq y_i} w_i^{(n)} + \exp(-\beta) \sum_{i=1}^{l} w_i^{(n)}$$

Hint: do it in the usual way (i.e. by differentiating the expression and setting it to 0).

Using the abbreviation $\mathrm{err}_n = \frac{\sum_{i=1}^{l} w_i^{(n)} I(y_i \neq b_n(\mathbf{x}_i))}{\sum_{i=1}^{l} w_i^{(n)}}$ show that the obtained expression is $\beta_n = \frac{1}{2}\ln\frac{1-\mathrm{err}_n}{\mathrm{err}_n}$.
Note that $\beta_n \geq 0$, by our assumption that all classifiers are better than random guessing, i.e. $\mathrm{err}_n \leq \frac{1}{2}$, so the result is in accordance with the previous subtask.

Please provide reasoning and explanations in full sentences. Grading of the task will heavily depend on it.

*Write your second step** calculation here*

1. Derivate ...
2. Using given abbreviation show ...

$$1.\ (\exp(\beta) - \exp(-\beta)) \sum_{b_n(\mathbf{x}_i)\neq y_i} w_i^{(n)} + \exp(-\beta) \sum_{i=1}^{l} w_i^{(n)}$$
$$= \exp(\beta) \sum_{b_n(\mathbf{x}_i)\neq y_i} w_i^{(n)} - \exp(-\beta) \sum_{b_n(\mathbf{x}_i)\neq y_i} w_i^{(n)} + \exp(-\beta) \sum_{i=1}^{l} w_i^{(n)}$$
$$\Rightarrow \frac{\partial}{\partial\beta}\Big(\exp(\beta) \sum_{b_n(\mathbf{x}_i)\neq y_i} w_i^{(n)} - \exp(-\beta) \sum_{b_n(\mathbf{x}_i)\neq y_i} w_i^{(n)} + \exp(-\beta) \sum_{i=1}^{l} w_i^{(n)}\Big)$$
$$= \exp(\beta) \sum_{b_n(\mathbf{x}_i)\neq y_i} w_i^{(n)} + \exp(-\beta) \sum_{b_n(\mathbf{x}_i)\neq y_i} w_i^{(n)} - \exp(-\beta) \sum_{i=1}^{l} w_i^{(n)}$$
$$= (\exp(\beta) + \exp(-\beta)) \sum_{b(\mathbf{x}_i)\neq y_i} w_i^{(n)} - \exp(-\beta) \sum_{i=1}^{l} w_i^{(n)}$$

$$2.\ (\exp(\beta) + \exp(-\beta)) \sum_{b_n(\mathbf{x}_i)\neq y_i} w_i^{(n)} - \exp(-\beta) \sum_{i=1}^{l} w_i^{(n)} = 0$$
$$\Leftrightarrow (\exp(\beta) + \exp(-\beta)) \sum_{b(\mathbf{x}_i)\neq y_i} w_i^{(n)} = \exp(-\beta) \sum_{i=1}^{l} w_i^{(n)} \mid \cdot \ln$$
$$\Leftrightarrow \ln\Big((\exp(\beta) + \exp(-\beta)) \sum_{b_n(\mathbf{x}_i)\neq y_i} w_i^{(n)}\Big) = \ln\Big(\exp(-\beta) \sum_{i=1}^{l} w_i^{(n)}\Big)$$
$$\Leftrightarrow \ln\big((\exp(\beta) + \exp(-\beta))\big) + \ln\Big(\sum_{b_n(\mathbf{x}_i)\neq y_i} w_i^{(n)}\Big) = \ln\big(\exp(-\beta)\big) + \ln\Big(\sum_{i=1}^{l} w_i^{(n)}\Big)$$
$$\Leftrightarrow \ln\big((\exp(\beta) + \exp(-\beta))\big) + \ln\Big(\sum_{i=1}^{l} w_i^{(n)} I(b_n(\mathbf{x}_i) \neq y_i)\Big) = -\beta + \ln\Big(\sum_{i=1}^{l} w_i^{(n)}\Big)$$
$$\Leftrightarrow \ln\big((\exp(\beta) + \exp(-\beta))\big) + \beta = \ln\Big(\sum_{i=1}^{l} w_i^{(n)}\Big) - \ln\Big(\sum_{i=1}^{l} w_i^{(n)} I(b_n(\mathbf{x}_i) \neq y_i)\Big)$$
$$\Leftrightarrow \ln\big((\exp(\beta) + \exp(-\beta))\big) + \ln(\exp(\beta)) = \ln\Big(\frac{\sum_{i=1}^{l} w_i^{(n)}}{\sum_{i=1}^{l} w_i^{(n)} I(b_n(\mathbf{x}_i) \neq y_i)}\Big)$$
$$\text{Since } \mathrm{err}_n = \frac{\sum_{i=1}^{l} w_i^{(n)} I(y_i \neq b_n(\mathbf{x}_i))}{\sum_{i=1}^{l} w_i^{(n)}}$$
$$\Rightarrow \frac{1}{\mathrm{err}_n} = \frac{\sum_{i=1}^{l} w_i^{(n)}}{\sum_{i=1}^{l} w_i^{(n)} I(y_i \neq b_n(\mathbf{x}_i))}$$
$$\Rightarrow \ln\big((\exp(\beta)^2 + \exp(-\beta)\exp(\beta))\big) = \ln\Big(\frac{1}{\mathrm{err}_n}\Big)$$
$$\Rightarrow \ln(\exp(\beta)^2 + 1) = \ln\Big(\frac{1}{\mathrm{err}_n}\Big)$$
$$\Rightarrow \exp(\beta)^2 + 1 = \frac{1}{\mathrm{err}_n}$$
$$\Rightarrow \exp(\beta)^2 = \frac{1 - \mathrm{err}_n}{\mathrm{err}_n} \mid \cdot \ln$$
$$\Rightarrow 2\beta = \ln\Big(\frac{1 - \mathrm{err}_n}{\mathrm{err}_n}\Big)$$
$$\Rightarrow \beta_n = \frac{1}{2}\ln\Big(\frac{1 - \mathrm{err}_n}{\mathrm{err}_n}\Big)$$

Calculation 3 (10 points):

In the **final step** we can update the approximation as follows: $g_n(\mathbf{x}) = g_{n-1}(\mathbf{x}) + \beta_n b_n(\mathbf{x})$. To finish the proof proceed by deriving the following relations:

1. The weights for the next generation can be computed as follows: $w_i^{(n+1)} = w_i^{(n)} \exp(-y_i \beta_n b_n(\mathbf{x}))$.
2. $-y_i b_n(\mathbf{x}) = 2I(y_i \neq b_n(\mathbf{x}_i)) - 1$
3. Use these two relations to show that:

$$w_i^{(n+1)} = w_i^{(n)} \exp(-\beta_n) \exp(\alpha_n I(y_i \neq b_n(\mathbf{x}_i))) \qquad (2)$$

where $\alpha_n = 2\beta_n$ is the α_n from the AdaBoostM1 algorithm from the lecture.

Please provide reasoning and explanations in full sentences. Grading of the task will heavily depend on it.

*Write your final step** calculation here*

1. Calculate weights ...
2. ...
3. ...

$$1.\ \text{As stated in the task description: } w_i^{(n)} = \exp(-y_i g_{n-1}(\mathbf{x}_i))$$
$$\Rightarrow w_i^{(n+1)} = \exp(-y_i g_n(\mathbf{x}_i))$$
$$\text{Using } g_n(\mathbf{x}) = g_{n-1}(\mathbf{x}) + \beta_n b_n(\mathbf{x}) \Rightarrow$$
$$= exp(-y_i g_{n-1}\mathbf{x}_i - y_i \beta_n b_n(\mathbf{x}))$$
$$= w_i^{(n)} \exp(-y_i \beta_n b_n(\mathbf{x})$$

$$2.\ \text{There are two possibilities: either the datapoint is correctly classified, either misclassified}$$
$$\text{I. If datapoint is correctly classified } \Rightarrow y_i b_n(\mathbf{x}_i) = 1 \mid \cdot(-1)$$
$$\Leftrightarrow -y_i b_n(\mathbf{x}_i) = -1$$
$$\text{if } y_i b_n(\mathbf{x}_i) = 1 \Rightarrow I(y_i \neq b_n(\mathbf{x}_i)) = 0 = 2I(y_i \neq b_n(\mathbf{x}_i))$$
$$\Rightarrow -y_i b_n(\mathbf{x}_i) = 2I(y_i \neq b_n(\mathbf{x}_i)) - 1 \text{ (q.e.d)}$$

$$\text{II. If datapoint is misclassified } \Rightarrow y_i b_n(\mathbf{x}_i) = -1 \mid \cdot(-1)$$
$$\Leftrightarrow -y_i b_n(\mathbf{x}_i) = 1$$
$$\text{if } y_i b_n(\mathbf{x}_i) = -1 \Rightarrow I(y_i \neq b_n(\mathbf{x}_i)) = 1 \Rightarrow 2I(y_i \neq b_n(\mathbf{x}_i)) = 2$$
$$\Rightarrow -y_i b_n(\mathbf{x}_i) = 2I(y_i \neq b_n(\mathbf{x}_i)) - 1 \text{ (q.e.d)}$$

$$\text{Thus } -y_i b_n(\mathbf{x}) = 2I(y_i \neq b_n(\mathbf{x}_i)) - 1 \text{ holds even though the datapoints are classified correctly or misclassifid}$$

$$3.\ \text{Using the formula from subtask 1: } w_i^{(n+1)} = w_i^{(n)} \exp(-y_i \beta_n b_n(\mathbf{x}))$$
$$\text{Using the formula from subtask 2: } -y_i b_n(\mathbf{x}) = 2I(y_i \neq b_n(\mathbf{x}_i)) - 1 \Rightarrow$$
$$w_i^{(n+1)} = w_i^{(n)} \exp(\beta_n(2I(y_i \neq b_n(\mathbf{x}_i)) - 1))$$
$$= w_i^{(n)} \exp(2\beta_n I(y_i \neq b_n(\mathbf{x}_i)) - \beta_n)$$
$$= w_i^{(n)} \exp(-\beta_n) \exp(2\beta_n I(y_i \neq b_n(\mathbf{x}_i)))$$
$$\text{with } 2\beta_n = \alpha_n \Rightarrow w_i^{(n+1)} = w_i^{(n)} \exp(-\beta_n) \exp(\alpha_n I(y_i \neq b_n(\mathbf{x}_i)))$$

Task 2: Random Forests and Feature importance

In this task you will train a Random Forest (RF) Classifier on a subset of fashionMNIST. You should observe how these models can immediately give you useful information about feature importance, which is a very convenient property of RFs.

First you should re-use the given code from the previous assignment to load the whole data set (the procedure is completely analogous).

- Next implement code that filters the data set for the classes with labels 1 (trousers) and 9 (ankle boots).You should create the filtered train data set from x_train and y_train and the test data set from x_test and y_test. **Hint**: Masks provide a convenient solution to this task.
- After the filtering procedure the data samples corresponding to trousers should be labelled as 1 and the ankle boots as 0. Perform this step on the test and train data set.
- To accomplish this task, implement a function **_filter_()**.

Code 1 (15 points):

In [1]:
```
#NOTE##################################################################################################
#Please add all your imports in this cell only
#######################################################################################################

#Nothing to do here
import numpy as np
import pandas as pd
import sys
import time
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from mnist_loader import MNIST
import matplotlib.pyplot as plt
from matplotlib import style
import matplotlib as mpl
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix
# Set random seed to ensure reproducible runs
RSEED = 10
```

In [2]:
```
#Load training and test data (routine from last week)
data = MNIST('./dataset/')
img_train, labels_train = data.load_training()
x_train = np.array(img_train)
y_train = np.array(labels_train)
x_test,y_test = data.load_testing()
x_test = np.array(x_test)
y_test = np.array(y_test)
print(y_train)
print(y_test)
```

```
[2 9 6 ... 8 8 7]
[0 1 2 ... 8 8 1]
```

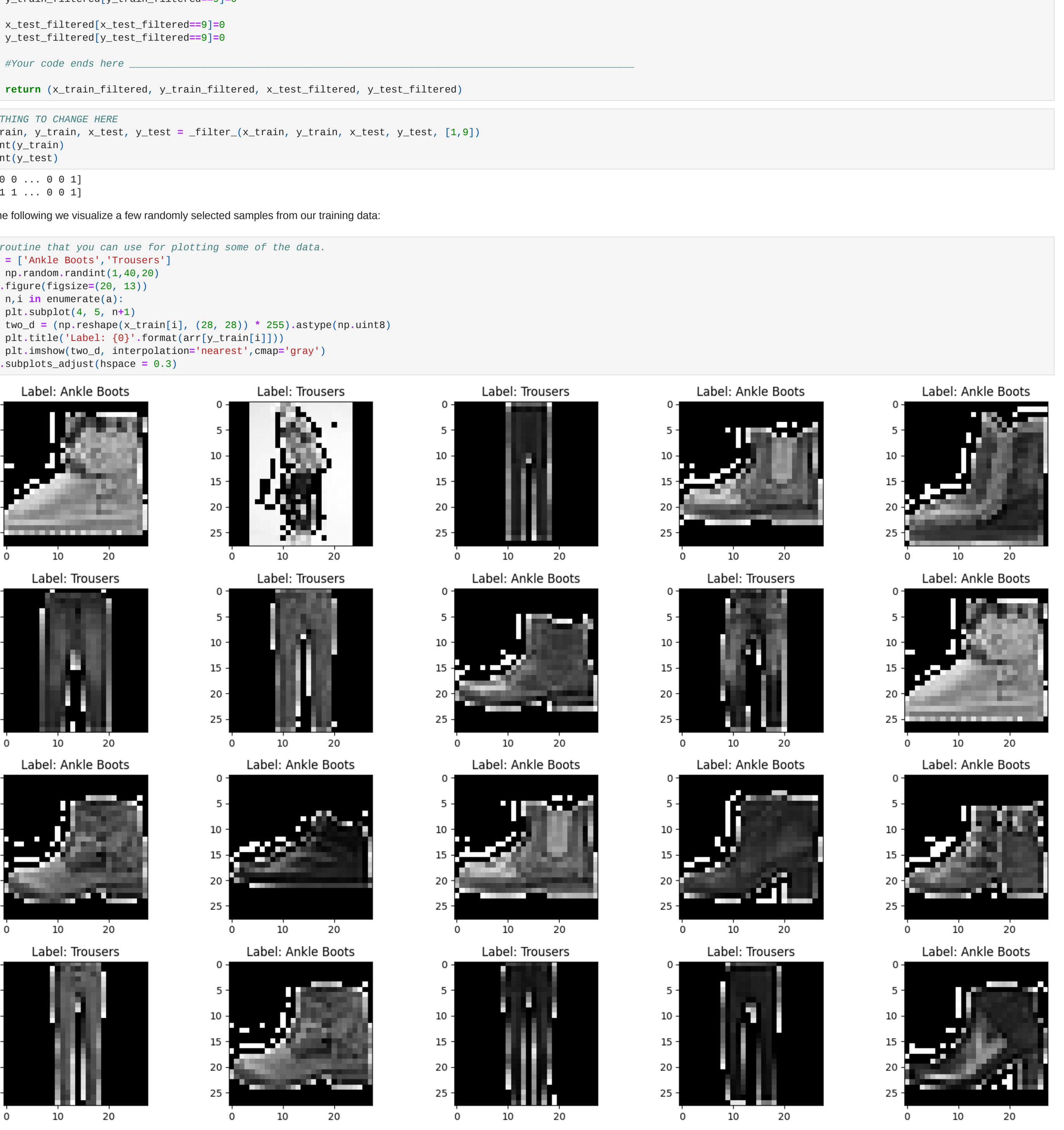
In [3]:
```
"""Function _filter_() is created to filter datasets w.r.t to given labels
@param x_train Training feature matrix
@param y_train Training labels vector
@param x_test Test feature matrix
@param y_test Test labels vector
@param labels_list list of length 2 which consists of integer labels.

@returns a tuple (x_train_filtered, y_train_filtered, x_test_filtered, y_test_filtered).
"""
def _filter_(x_train:np.array,
             y_train:np.array,
             x_test:np.array,
             y_test:np.array,
             labels_list:list):

    #Your code goes here ↓↓↓

    label1=labels_list[0]
    label2=labels_list[1]


    x_train_filtered = x_train[np.where(np.logical_or(y_train==label1,y_train==label2))]
    y_train_filtered = y_train[np.where(np.logical_or(y_train==label1,y_train==label2))]

    x_test_filtered = x_test[np.where(np.logical_or(y_test==label1,y_test==label2))]
    y_test_filtered = y_test[np.where(np.logical_or(y_test==label1,y_test==label2))]

    x_train_filtered[x_train_filtered==9]=0
    y_train_filtered[y_train_filtered==9]=0

    x_test_filtered[x_test_filtered==9]=0
    y_test_filtered[y_test_filtered==9]=0

    #Your code ends here ___________________________________________________________________________

    return (x_train_filtered, y_train_filtered, x_test_filtered, y_test_filtered)
```

In [4]:
```
#NOTHING TO CHANGE HERE
x_train, y_train, x_test, y_test = _filter_(x_train, y_train, x_test, y_test, [1,9])
print(y_train)
print(y_test)
```

```
[0 0 0 ... 0 0 1]
[1 1 1 ... 0 0 1]
```

In the following we visualize a few randomly selected samples from our training data:

In [5]:
```
#A routine that you can use for plotting some of the data.
arr = ['Ankle Boots','Trousers']
a = np.random.randint(1,40,20)
plt.figure(figsize=(20, 13))
for n,i in enumerate(a):
    plt.subplot(4, 5, n+1)
    two_d = (np.reshape(x_train[i], (28, 28)) * 255).astype(np.uint8)
    plt.title('Label: {0}'.format(arr[y_train[i]]))
    plt.imshow(two_d, interpolation='nearest',cmap='gray')
plt.subplots_adjust(hspace = 0.3)
```



Code (5 points):

Your task now is to train a sklearn RandomForestClassifier with the default parameters on the training data set.

Then get the model's predictions for the test data set. Use **RSEED** as random_seed for the RandomForestClassifier.

For this, we ask you to implement a function fit_predict.

In [6]:
```
"""Function fit_predict() is created to fit RF on training data and return predictions as well as model
@param x_train Training feature matrix
@param y_train Training labels vector
@param x_test Test feature matrix
@param y_test Test labels vector
@param rseed Random Seed, integer

@returns a tuple (model, prediction). Where model is an trained RF classifier and prediction is a np array
"""
def fit_predict(x_train,y_train,x_test,y_test, rseed):

    #Your code goes here ↓↓↓
    model = RandomForestClassifier(random_state=rseed)
    model.fit(x_train,y_train)

    prediction = model.predict(x_test)
    #Your code ends here ___________________________________________________________________________

    return model, prediction
```

In [7]:
```
model, prediction = fit_predict(x_train,y_train,x_test,y_test, RSEED)
```

Code (15 points):

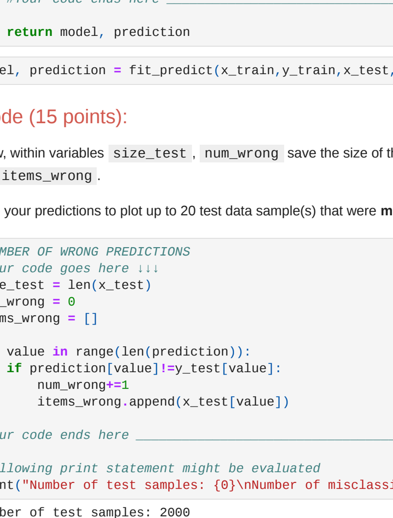
Now, within variables `size_test`, `num_wrong` save the size of the test set and number of misclassified test samples. Both variables should be integers. Retrieve misclassified samples from test set and save them as a list `items_wrong`.

Use your predictions to plot up to 20 test data sample(s) that were **missclassified**:

In [8]:
```
#NUMBER OF WRONG PREDICTIONS
#Your code goes here ↓↓↓
size_test = len(x_test)
num_wrong = 0
items_wrong = []

for value in range(len(prediction)):
    if prediction[value]!=y_test[value]:
        num_wrong+=1
        items_wrong.append(x_test[value])

#Your code ends here ___________________________________________________________________________

#Following print statement might be evaluated
print("Number of test samples: {0}\nNumber of misclassified samples: {1}".format(size_test,num_wrong))
```

```
Number of test samples: 2000
Number of misclassified samples: 1
```

In [9]:
```
#PLOTTING WRONG PREDICTIONS

#Your code goes here ↓↓↓

#Use your freestlye plotting, or routine we have shown above
up_to_no_samples=range(20)
for i in up_to_no_samples:
    if i>=len(items_wrong):
        break
    fig=plt.figure
    two_d = (np.reshape(items_wrong[i], (28, 28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest',cmap='gray')

plt.show()
#Your code ends here ___________________________________________________________________________
```



Code (20 points):

Within this part we will try to see the decision-making incentives of Random Forest.

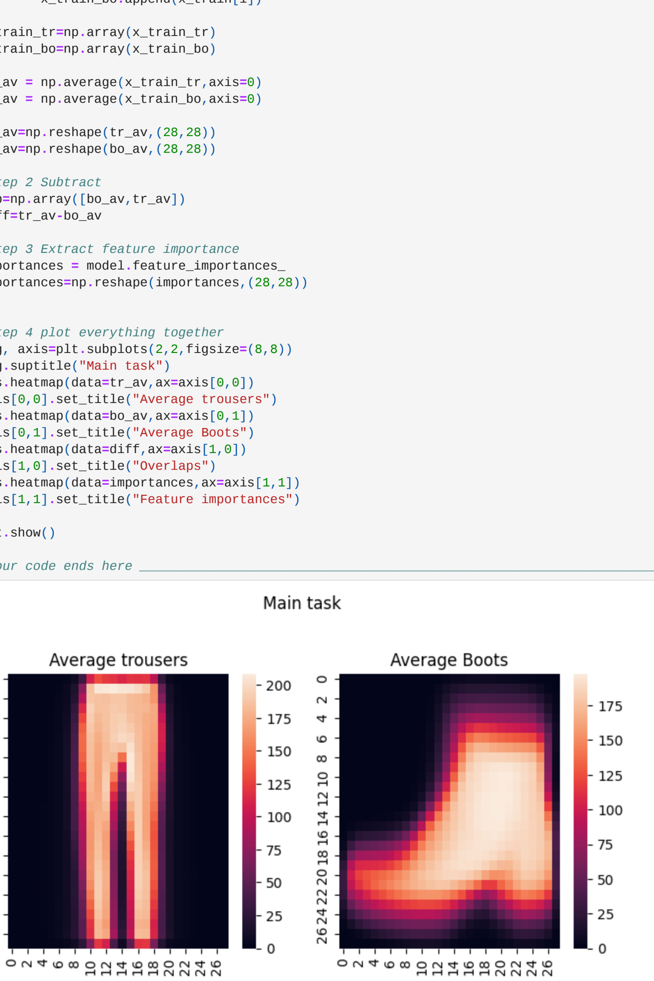
To do this we ask you to implement the following tasks:

1. Take your training dataset and split it into 2: trousers and boots.
   Calculate the average of features. Reshape averages the to 2D arrays of shape 28*28 and plot them as heatmaps. Save results under variables **tr_av** and **bo_av**
2. From the average of trousers subtract the average of boots. Save result as a variable **diff**. Plot it as a heatmap.
3. Define the feature importance of the previously trained RF classifier witin variable **importances**. Visualize it as a heatmap.
   Hint: Check scikit-learn documentation to access feature importance.

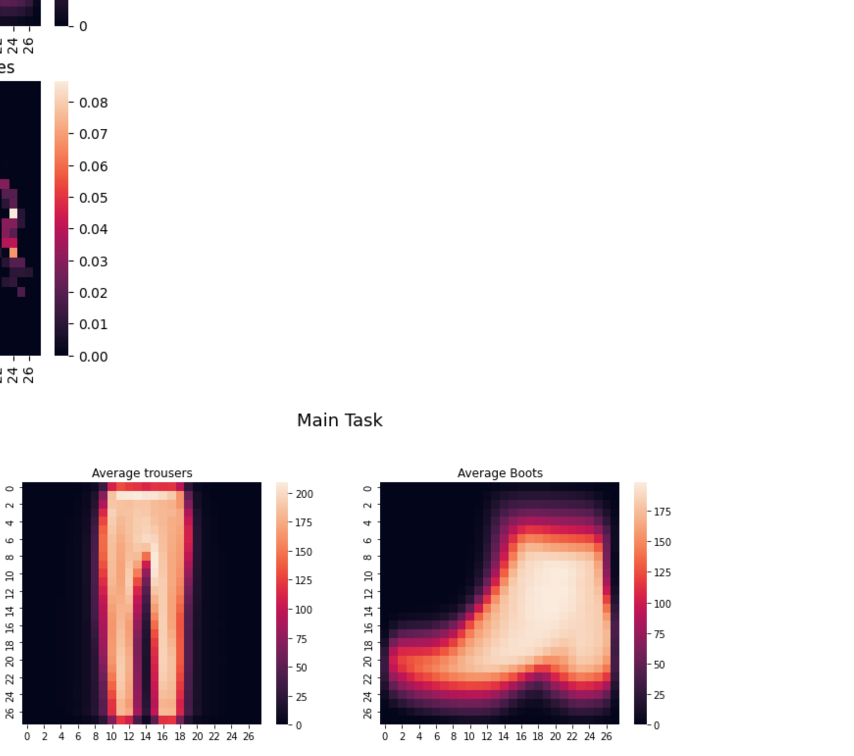
The evaluation of the following code will be done by viewing your plots.

For those who are curious: run RF under different seeds, and look how plots are changing.
BEFORE SUBMISSION RETURN TO ORIGINAL SEED = 10

In [10]:
```
#PLOTTING HEATMAPS

#Your code goes here ↓↓↓

#Use your freestlye plotting
#No plots' variables are needed

#Step 1 Split and calculate averages
x_train_tr,x_train_bo=[],[]
for i in range(len(x_train)):
    if y_train[i]==1:
        x_train_tr.append(x_train[i])
    if y_train[i]==0:
        x_train_bo.append(x_train[i])

x_train_tr=np.array(x_train_tr)
x_train_bo=np.array(x_train_bo)

tr_av = np.average(x_train_tr,axis=0)
bo_av = np.average(x_train_bo,axis=0)

tr_av=np.reshape(tr_av,(28,28))
bo_av=np.reshape(bo_av,(28,28))

#Step 2 Subtract
tmp=np.array([bo_av,tr_av])
diff=tr_av-bo_av

#Step 3 Extract feature importance
importances = model.feature_importances_
importances=np.reshape(importances,(28,28))


#Step 4 plot everything together
fig, axis=plt.subplots(2,2,figsize=(8,8))
fig.suptitle("Main task")
sns.heatmap(data=tr_av,ax=axis[0,0])
axis[0,0].set_title("Average trousers")
sns.heatmap(data=bo_av,ax=axis[0,1])
axis[0,1].set_title("Average Boots")
sns.heatmap(data=diff,ax=axis[1,0])
axis[1,0].set_title("Overlaps")
sns.heatmap(data=importances,ax=axis[1,1])
axis[1,1].set_title("Feature importances")

plt.show()

#Your code ends here ___________________________________________________________________________
```



If you have solved the previous task correctly, the resulting plot should look close to this:



Question (5 points):

What observations can you make?
(Multiple answers might be correct)

To answer the question assign to variables in the nex cell **True** or **False** boolean values. To earn points **assign values to all variables**.
NOTE Do not reuse these variable names. They are used for testing.

a_) RF achieves accuracy lower than 85%
b_) The most important features are located in image regions that primarily show either only trousers or ankle boots as depicted by the plots of average trousers and boots.
c_) If one pixel would always be bright for trousers but never for ankle boots the RF classifier would certainly learn to use only this pixel as one of the features.
d_) Comparing the misclassified ankle boot to other ankle boots suggests that it might be an outlier in terms of background brightness.

In [11]:
```
#examples for you
example_of_true_variable = True
example_of_false_variable = False

#your answers go here ↓↓↓
a_=False
b_=True
c_=False
d_=True
```

In [12]:
```
def filter_tl(xt,yt,xte,yte,labels_list):
    try:
        xtf, ytf, xtef, ytef = _filter_(xt,yt,xte,yte,labels_list)
        res = (xtf.tolist(), ytf.tolist(), xtef.tolist(), ytef.tolist())
        print('Filtering works')
        return res
    except Exception as e:
        print("Execution of _filter_, or convertion from np.array to list failed")
        print(e)
        raise ValueError()

def model_param(xt,yt,xte,yte):
    try:
        model, pred = fit_predict(xt,yt,xte,yte,RSEED)
        nfeat = model.n_features_in_
        nclass = model.n_classes_
        print("Model is valid object")
        return nfeat,nclass
    except Exception as e:
        print("Execution of fit_predict, or model parameters extraction failed")
        print(e)
        raise ValueError()

def testoptions(options):
    for elem in options:
        if elem!=True and elem!=False and elem!=None:
            raise ValueError(f"Check answers for questions again")
    print("Test questions answers are ok")
```

In [13]:
```
#these are dummy inputs, your code should be runnable on such
xt,yt,xte,yte = np.array([[1,1,1],[9,9,9],[5,5,5]]),np.array([1,9,5]),np.array([[0,0,0],[1,1,1],[9,9,9]]),np.array([0,1,9])

xt,yt,xte,yte = filter_tl(xt,yt,xte,yte,[1,9])
model_param(xt,yt,xte,yte)
testoptions(np.array([a_,b_,c_,d_]))
print('Executable')
```

```
Filtering works
Model is valid object
Test questions answers are ok
Executable
```