Benedikt Hopf
Alireza Ketabdari

1	2	3	4	Σ

Exercise Sheet Nr. 2

(Deadline November 23, 2021)

Problem 1

a) A square symmetric matrix $H \in R^{n \times n}$ is positive semi-definite (*PSD*) for any vector v with real components if

$$(v^T H v \geq 0) \qquad (\forall v \in \mathbb{R}^n)$$

in other word, the dot product of Hv and v is nonnegative,

$$\langle v, Hv \rangle \geq 0 \qquad (\forall v \in \mathbb{R}^n)$$

In geometric terms, the condition of positive semidefiniteness says that, for every v, the angle between v and Hv does not exceed $\frac{\pi}{2}$. Indeed, $0 \leq \langle Hv, v \rangle = \|Hv\|\|x\| \cos\theta$ and so $\cos\theta \geq 0$.

EXAMPLE: Let $H = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$. Then $Hv = \begin{pmatrix} x_1 \\ 2x_2 \end{pmatrix}$ and $\langle Hv, v \rangle = x_1^2 + x_2^2 \geq 0$ implying that A is positive semidefinite.

b) The so-called *weighted degree (WD) kernel* computes similarities between sequences while taking positional information of multiple *k-mers* into account. The main idea of the *WD kernel* is to count the (exact) co-occurrences of *k-mers* at corresponding positions in the two sequences to be compared.

The recognition of matching blocks using the *WD kernel* strongly depends on the position of the sub-sequence and does not tolerate any positional variation. For instance, if a consecutive block in one sequence is shifted by only one position, the *WD kernel* fails to discover similar blocks and returns a lower similarity score. Depending on the application in mind, this problem might lead to suboptimal results. Hence the *WD kernel with shifts (WDS)*, which shifts the two sequences against each other in order to tolerate a small positional variations of sequence motifs. Conceptually, it is a mixture between the *WD* and the *spectrum kernel*.

The duty of these two kernels are computing similarities between sequences. When we are only comparing every single letter (for example the i_{th} letter) from the first sequence with exactly the corresponding letter in second sequence (so also the i_{th} letter), this is called *WD without shifts*. On the other hand, if we can also compare the letter in first sequence (for example the i_{th} letter) with a shifted letter in the second sequence (for example the $i - 1_{th}$ letter), this is the *WD kernel with shifts*. Our sequences can be two different *SNPs*.

c) In the process of splicing, different parts of a gene may be included within (*Exon*) or excluded *(Intron)* from the final processed messenger *RNA (mRNA)* produced from that gene. This splicing can be performed in different ways, resluting in different exons joined together. This means the *exons* are joined in different combinations, leading to different (alternative) *mRNA* strands. Consequently, the proteins translated from alternatively spliced *mRNAs* will contain differences in their amino acid

sequence and, often, in their biological functions. Notably, alternative splicing allows the human genome to direct the synthesis of many more proteins ($\approx 100,000$) than would be expected from its 24,000 protein-coding genes.

When transcribing the *pre-mRNA* to *mRNA*, the introns are left out and only the extrons included. This means, that the exons, have to be put together in one sequence again. When two exons are connected together, the right side on the left exon is called the *acceptor site* and the left side of the right exon is called the *donor side*.

d) *Domain adaptation* is the ability to apply an algorithm trained in one or more *source domains* (combination of an input space X, an output space Y and an associated probability distribution P) to a different (but related) *target domain*. This is useful, if training data is rare and/or expansive to get. This way a learning algorithm, can (in addition to the actual training data) also learn from other data (e.g. from a different animal), extending the amount of training data available. How good this approach works depends strongly on the similarity of the domains. If the domains are baically the same it works as well as having only the actual training data. If the domains are not similar at all, it can even be detremental.

   a) One way to do domain adaptation is to do a convex combination of the results of two classifiers, where one has been trained on the target domain (the *actual* domain), and the other one on the source domain (the supplementary domain):

$$classifier_{total} = \alpha \cdot classifier_{target} + (1 - \alpha) classifier_{source} \quad \alpha \in [0, 1]$$

   b) Another option is to train the classifier on the joint data, but penalize errors on the target domain stronger than on the soruce domain. So the total loss L_{total} would look like this:

$$L_{total} = \alpha \cdot L_{target} + \beta \cdot L_{source} \quad \alpha, \beta \in \mathbb{R}$$

e) Multitask learning describes a setting where one has several (similar) tasks, some data for the tasks and information about the relationship between the tasks (i.e. a tree encoding the similarity). The goal is to construct a learning model for each of the tasks, which uses all of the data in some way. This is similar to domain adaptation. One way to do this is to first learn a model for the entire data, and then learn more and more special models for the individual tasks, while using the general model as a regularizer.

   Whether multitask learning with many source domains or dualtask learning performs better depends on the amount of training data per task and the similarity of the tasks. If there is very little training data dualtask learning will deliver bad performance, as there is just to little data, to properly learn from. If the tasks are very different, multitask learning is not going to perform well, since the other tasks are too unrelated to the traget task, and therefore the aditional data will just distract the model. On the other hand, dual task learning (assuming that is is done on the target and the most similar task) will not have that problem (as much). Multitask learning can be good, if there is little data, since it allows using data from related tasks, and therefore increase the amount of available data to a more usable amount.

f) The main difference between the two groups is that *MHC* is the general term, whereas *HLA* is only used for human *MHC*-molecules.

MHC (HLA) class I molecules are used to to take peptides (usually of length 9) from borken up proteins found inside a cell to the outside and present them to T-cells, thereby triggering an immune reaction against these peptides.

g) *Leveraging* is the process of encoding data in such a way, that it can be shared between tasks. So instead of just using information about the *HLA* one would also input additional information, such as chemical properties or *HLA*-supertypes. This enables one single model to learn several tasks, while using all the available training data. At inference time the data is the input together with other known properties, enabling the model to do inference with the help of these additional properties (and the molecules with such properties that is has seen during training).

Problem 2

Proof.

$$\begin{aligned}
\|\phi(x_i) - \phi(x_j)\|^2 &= \langle \phi(x_i) - \phi(x_j), \phi(x_i) - \phi(x_j) \rangle \\
&= \langle \phi(x_i), \phi(x_i) - \phi(x_j) \rangle - \langle \phi(x_j), \phi(x_i) - \phi(x_j) \rangle \\
&= \langle \phi(x_i), \phi(x_i) \rangle - \langle \phi(x_i), \phi(x_j) \rangle - \langle \phi(x_j), \phi(x_i) \rangle + \langle \phi(x_j), \phi(x_j) \rangle \\
&= \langle \phi(x_i), \phi(x_i) \rangle - \langle \phi(x_i), \phi(x_j) \rangle - \langle \phi(x_i), \phi(x_j) \rangle + \langle \phi(x_j), \phi(x_j) \rangle \\
&= \langle \phi(x_i), \phi(x_i) \rangle - 2 \cdot \langle \phi(x_i), \phi(x_j) \rangle + \langle \phi(x_j), \phi(x_j) \rangle \\
&= k(x_i, x_i) - 2 \cdot k(x_i, x_j) + k(x_j, x_j)
\end{aligned}$$

□

Problem 3

See next page (jupyter).

Problem3

November 18, 2021

1 Problem 3

[22]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[9]:
```
indices = []
strings = []
with open("sequencesMSAfasta", "r") as file:
    for line in file.readlines():
        if line.startswith(">"): indices.append(line[:-1])
        else: strings.append(line[:-1])
```

[14]:
```
data = pd.DataFrame(
    data=strings,
    index=indices,
    columns=["String"]
)
data.head()
```

[14]:

	String
>AF153142	TGTACAAGACCCAACAATAATACAAGAAAAAGTATAAGGATAGGAC…
>AF153143	TGTACAAGGCCCGGCAATAATACAAGGAAAAGTATGAGGATAGGAC…
>AF153144	TGTACAAGACCCAATAATAATACAAGAAAAAGCATAAGGATAGGAC…
>AF153145	TGTACAAGACCCAACAATAATACAAGAAAAAGTATAAGGATAGGAC…
>HQ906866	TGCACAAGGCCCTACGATAAGGTAAGCTACAGGACACCTATAGGAR…

[44]:
```
def wdk(s1, s2, d=None, beta=None):
    assert len(s1) == len(s2)
    L = len(s1)

    if beta is None:
        beta = [1 for _ in s1]
    if d is None:
        d = len(beta)

    assert d <= len(beta)
```

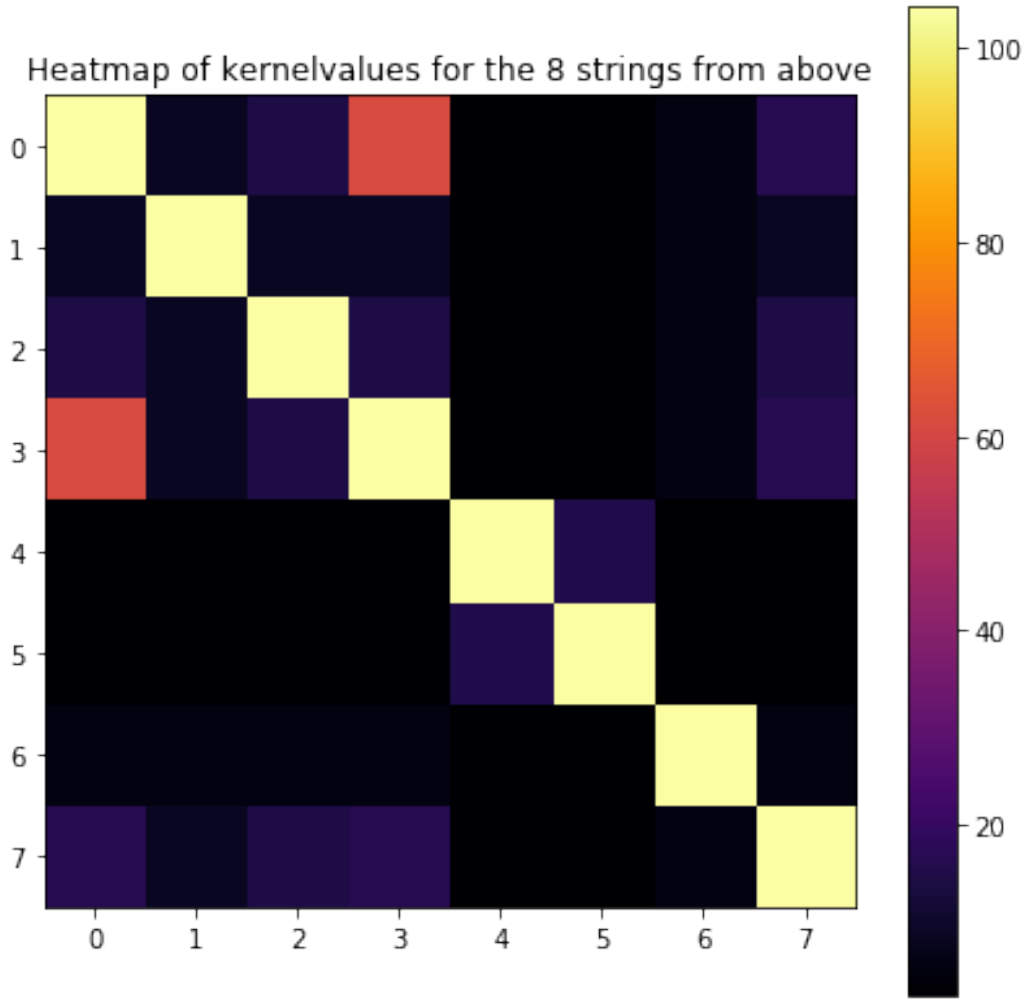
```
    return np.concatenate([[
        beta[k] * (1 if s1[k:k+l] == s2[k:k+l] else 0)
    for l in range(L-k)]
    for k in range(d)]).sum()
```

[29]:
```
def kernel(s, f, **kwargs):
    return np.array([[
        wdk(x, y, **kwargs)
    for x in s]
    for y in s])
```

[30]:
```
d = 3
beta = [
    2 * (d - k + 1) / (d*(d+1))
for k in range(1, d+1)]
```

[62]:
```
kernel_matrix = kernel(data.String, wdk, d=d, beta=beta)

plt.figure(figsize=(7, 7))
plt.imshow(kernel_matrix, cmap="inferno")
plt.colorbar()
plt.title("Heatmap of kernelvalues for the 8 strings from above")
plt.show()
```



As one can see, the largest values are on the diagonal, which makes sense, since these are the entries, where the string has been compared to itself.

Problem4

November 23, 2021

1 Problem 4

[1]:
```
from sklearn import svm
from sklearn.model_selection import KFold
from sklearn.metrics import roc_curve, roc_auc_score
import pandas as pd
import numpy as np
from concurrent.futures import ProcessPoolExecutor
import matplotlib.pyplot as plt
```

1.1 a) Implement Kernels

[2]:
```
def k_dirac(x, y):
    return 1 if x == y else 0
def k_uniform(x, y):
    return 1
def k_multitask(x, y, lmbd=0.5):
    return lmbd * k_dirac(x, y) + (1-lmbd) * k_uniform(x, y)
def k_linseq(x, y):
    return (np.array(list(x)) == np.array(list(y))).sum()
def k_poly(x, y, d=2):
    return (1 + k_linseq(x, y))**d
def join_kernel_matricies(*kernels):
    return np.array(kernels).prod(axis=0)
```

[3]:
```
def kernel(k):
    return lambda X, Y: np.array([[
        k(x, y)
    for y in Y]
    for x in X])
```

1.2 b) Build SVM and do cross-validation

[4]:
```
binding_data = pd.read_csv("BindingData.csv")
```

[5]:
```
binding_data.head()
```

[5]:
```
   Epitope ID  binding information       allele  peptide sequence  \
0       52886             Positive  HLA-A*01:01         QYDPVAALF
1       71650             Positive  HLA-A*01:01         VVEKQSGLY
2      150319             Positive  HLA-A*01:01         ITEAELTGY
3      141221             Positive  HLA-A*01:01         ATDSLNNEY
4       24819             Positive  HLA-A*01:01         HSNLNDATY

    binding label
0               1
1               1
2               1
3               1
4               1
```

[6]:
```
pa_data = binding_data[[" allele", " peptide sequence"]].to_numpy()
print(pa_data)
```

```
[['HLA-A*01:01' 'QYDPVAALF']
 ['HLA-A*01:01' 'VVEKQSGLY']
 ['HLA-A*01:01' 'ITEAELTGY']
 …
 ['HLA-B*40:01' 'SEVKTLSSY']
 ['HLA-B*40:01' 'IELYSLAKY']
 ['HLA-B*40:01' 'KEGKLVIIF']]
```

[7]:
```
label_data = binding_data[" binding label"].to_numpy()
print(label_data)
```

```
[ 1  1  1 … -1 -1 -1]
```

[8]:
```
Cs = np.logspace(-4, 4, 9)
print(Cs)
```

```
[1.e-04 1.e-03 1.e-02 1.e-01 1.e+00 1.e+01 1.e+02 1.e+03 1.e+04]
```

[9]:
```
def do_cv(C):
    kf = KFold(n_splits=10, shuffle=True)

    scores = dict()
    names = ["uniform", "dirac", "multitask"]
    for name in names:
        scores[name] = {
            "accuracy": [],
            "results": [],
            "truths": [],
            "classifiers": []
        }
```

```
    for train_indices, test_indices in kf.split(pa_data):



        Xtrain = pa_data[train_indices]
        Ytrain = label_data[train_indices]
        Xtest = pa_data[test_indices]
        Ytest = label_data[test_indices]

        K_train_p = kernel(k_linseq)(Xtrain[:, 0], Xtrain[:, 0])
        K_test_p = kernel(k_linseq)(Xtest[:, 0], Xtrain[:, 0])

        for n, k in zip(
                        names,
                        [k_uniform, k_dirac, k_multitask]
        ):
            e = svm.SVC(
                kernel="precomputed",
                C=C
            )
            K_train_a = kernel(k)(Xtrain[:, 1], Xtrain[:, 1])
            K_test_a = kernel(k)(Xtest[:, 1], Xtrain[:, 1])

            Ktrain = join_kernel_matricies(K_train_p, K_train_a)
            Ktest = join_kernel_matricies(K_test_p, K_test_a)

            e.fit(Ktrain, Ytrain)
            score = e.score(Ktest, Ytest)
            scores[n]["accuracy"].append(score)
            print(f"Model {n} (C={C}): accuracy = {score:2.2f}")

            results = e.decision_function(Ktest)
            scores[n]["results"].append(results)
            scores[n]["truths"].append(Ytest)

            scores[n]["classifiers"].append(e)
    for n in scores.keys():
        scores[n]["accuracy"] = np.array(scores[n]["accuracy"]).mean()

    return scores
        #for name in names:
            #out.append(f"Mean accuracy for {name}-kernel and C={C}: {np.
↪array(scores[name]).mean()}")
```

[10]:
```
futures = []
with ProcessPoolExecutor() as ppe:
    for C in Cs:
```

```
        futures.append((C, ppe.submit(do_cv, C)))
scores = dict()
for C, future in futures:
    scores[C] = future.result()
```

```
Model uniform (C=1000.0): accuracy = 0.42
Model uniform (C=100.0): accuracy = 0.42
Model uniform (C=0.0001): accuracy = 0.52
Model uniform (C=10.0): accuracy = 0.42
Model uniform (C=0.1): accuracy = 0.48
Model uniform (C=0.01): accuracy = 0.52
Model uniform (C=0.001): accuracy = 0.52
Model dirac (C=1000.0): accuracy = 0.52
Model dirac (C=100.0): accuracy = 0.52
Model dirac (C=0.0001): accuracy = 0.52
Model uniform (C=1.0): accuracy = 0.42
Model dirac (C=10.0): accuracy = 0.52
Model dirac (C=0.001): accuracy = 0.52
Model dirac (C=0.01): accuracy = 0.52
Model dirac (C=0.1): accuracy = 0.52
Model dirac (C=1.0): accuracy = 0.52
Model multitask (C=1000.0): accuracy = 0.56
Model multitask (C=0.0001): accuracy = 0.52
Model multitask (C=100.0): accuracy = 0.56
Model multitask (C=10.0): accuracy = 0.56
Model multitask (C=0.001): accuracy = 0.52
Model multitask (C=0.1): accuracy = 0.46
Model multitask (C=0.01): accuracy = 0.52
Model multitask (C=1.0): accuracy = 0.56
Model uniform (C=0.0001): accuracy = 0.51
Model uniform (C=1000.0): accuracy = 0.44
Model uniform (C=100.0): accuracy = 0.44
Model uniform (C=0.1): accuracy = 0.46
Model dirac (C=0.0001): accuracy = 0.51
Model dirac (C=1000.0): accuracy = 0.51
Model uniform (C=0.001): accuracy = 0.51
Model dirac (C=100.0): accuracy = 0.51
Model uniform (C=10.0): accuracy = 0.44
Model uniform (C=0.01): accuracy = 0.51
Model dirac (C=0.1): accuracy = 0.51
Model dirac (C=0.001): accuracy = 0.51
Model uniform (C=1.0): accuracy = 0.44
Model dirac (C=10.0): accuracy = 0.51
Model dirac (C=0.01): accuracy = 0.51
Model multitask (C=0.0001): accuracy = 0.51
Model dirac (C=1.0): accuracy = 0.51
Model multitask (C=1000.0): accuracy = 0.63
```

```
Model multitask (C=100.0): accuracy = 0.63
Model multitask (C=0.1): accuracy = 0.52
Model multitask (C=0.001): accuracy = 0.51
Model multitask (C=0.01): accuracy = 0.51
Model multitask (C=10.0): accuracy = 0.63
Model multitask (C=1.0): accuracy = 0.63
Model uniform (C=1000.0): accuracy = 0.48
Model uniform (C=100.0): accuracy = 0.48
Model uniform (C=0.1): accuracy = 0.50
Model dirac (C=1000.0): accuracy = 0.55
Model uniform (C=0.0001): accuracy = 0.55
Model uniform (C=0.001): accuracy = 0.55
Model dirac (C=100.0): accuracy = 0.55
Model dirac (C=0.1): accuracy = 0.55
Model uniform (C=1.0): accuracy = 0.48
Model dirac (C=0.0001): accuracy = 0.55
Model uniform (C=0.01): accuracy = 0.55
Model uniform (C=10.0): accuracy = 0.48
Model dirac (C=0.001): accuracy = 0.55
Model dirac (C=1.0): accuracy = 0.55
Model multitask (C=1000.0): accuracy = 0.58
Model dirac (C=0.01): accuracy = 0.55
Model dirac (C=10.0): accuracy = 0.55
Model multitask (C=100.0): accuracy = 0.58
Model multitask (C=0.1): accuracy = 0.55
Model multitask (C=0.0001): accuracy = 0.55
Model multitask (C=0.001): accuracy = 0.55
Model multitask (C=1.0): accuracy = 0.58
Model multitask (C=0.01): accuracy = 0.55
Model multitask (C=10.0): accuracy = 0.58
Model uniform (C=1000.0): accuracy = 0.40
Model uniform (C=100.0): accuracy = 0.40
Model dirac (C=1000.0): accuracy = 0.52
Model uniform (C=0.1): accuracy = 0.45
Model uniform (C=0.0001): accuracy = 0.52
Model dirac (C=100.0): accuracy = 0.52
Model uniform (C=0.001): accuracy = 0.52
Model uniform (C=1.0): accuracy = 0.40
Model uniform (C=0.01): accuracy = 0.52
Model dirac (C=0.1): accuracy = 0.52
Model dirac (C=0.0001): accuracy = 0.52
Model dirac (C=0.001): accuracy = 0.52
Model multitask (C=1000.0): accuracy = 0.48
Model dirac (C=1.0): accuracy = 0.52
Model uniform (C=10.0): accuracy = 0.40
Model dirac (C=0.01): accuracy = 0.52
Model multitask (C=100.0): accuracy = 0.48
Model dirac (C=10.0): accuracy = 0.52
```

```
Model multitask (C=0.1): accuracy = 0.44
Model multitask (C=0.0001): accuracy = 0.52
Model multitask (C=0.001): accuracy = 0.52
Model multitask (C=1.0): accuracy = 0.48
Model multitask (C=0.01): accuracy = 0.52
Model multitask (C=10.0): accuracy = 0.48
Model uniform (C=1000.0): accuracy = 0.45
Model uniform (C=0.1): accuracy = 0.48
Model dirac (C=1000.0): accuracy = 0.52
Model uniform (C=100.0): accuracy = 0.45
Model uniform (C=0.0001): accuracy = 0.52
Model dirac (C=0.1): accuracy = 0.52
Model uniform (C=0.001): accuracy = 0.52
Model dirac (C=100.0): accuracy = 0.52
Model uniform (C=0.01): accuracy = 0.52
Model dirac (C=0.0001): accuracy = 0.52
Model multitask (C=1000.0): accuracy = 0.61
Model dirac (C=0.001): accuracy = 0.52
Model uniform (C=1.0): accuracy = 0.45
Model dirac (C=0.01): accuracy = 0.52
Model uniform (C=10.0): accuracy = 0.45
Model multitask (C=0.1): accuracy = 0.58
Model multitask (C=100.0): accuracy = 0.61
Model dirac (C=1.0): accuracy = 0.52
Model dirac (C=10.0): accuracy = 0.52
Model multitask (C=0.0001): accuracy = 0.52
Model multitask (C=0.001): accuracy = 0.52
Model multitask (C=0.01): accuracy = 0.52
Model multitask (C=1.0): accuracy = 0.61
Model multitask (C=10.0): accuracy = 0.61
Model uniform (C=1000.0): accuracy = 0.44
Model uniform (C=0.1): accuracy = 0.46
Model uniform (C=0.0001): accuracy = 0.50
Model dirac (C=1000.0): accuracy = 0.51
Model uniform (C=100.0): accuracy = 0.44
Model dirac (C=0.0001): accuracy = 0.50
Model dirac (C=0.1): accuracy = 0.51
Model uniform (C=0.01): accuracy = 0.50
Model uniform (C=0.001): accuracy = 0.50
Model dirac (C=100.0): accuracy = 0.51
Model dirac (C=0.01): accuracy = 0.50
Model dirac (C=0.001): accuracy = 0.50
Model multitask (C=1000.0): accuracy = 0.51
Model uniform (C=1.0): accuracy = 0.44
Model multitask (C=0.0001): accuracy = 0.50
Model multitask (C=0.1): accuracy = 0.49
Model uniform (C=10.0): accuracy = 0.44
Model dirac (C=1.0): accuracy = 0.51
```

```
Model multitask (C=100.0): accuracy = 0.51
Model multitask (C=0.01): accuracy = 0.50
Model multitask (C=0.001): accuracy = 0.50
Model dirac (C=10.0): accuracy = 0.51
Model multitask (C=1.0): accuracy = 0.51
Model multitask (C=10.0): accuracy = 0.51
Model uniform (C=1000.0): accuracy = 0.45
Model dirac (C=1000.0): accuracy = 0.56
Model uniform (C=0.1): accuracy = 0.50
Model uniform (C=0.0001): accuracy = 0.55
Model dirac (C=0.1): accuracy = 0.56
Model uniform (C=0.01): accuracy = 0.55
Model uniform (C=100.0): accuracy = 0.45
Model uniform (C=0.001): accuracy = 0.55
Model dirac (C=0.0001): accuracy = 0.55
Model multitask (C=1000.0): accuracy = 0.55
Model dirac (C=0.01): accuracy = 0.55
Model dirac (C=0.001): accuracy = 0.55
Model dirac (C=100.0): accuracy = 0.56
Model uniform (C=1.0): accuracy = 0.46
Model uniform (C=10.0): accuracy = 0.45
Model multitask (C=0.1): accuracy = 0.55
Model multitask (C=0.0001): accuracy = 0.55
Model dirac (C=1.0): accuracy = 0.56
Model dirac (C=10.0): accuracy = 0.56
Model multitask (C=0.01): accuracy = 0.55
Model multitask (C=0.001): accuracy = 0.55
Model multitask (C=100.0): accuracy = 0.55
Model multitask (C=1.0): accuracy = 0.55
Model multitask (C=10.0): accuracy = 0.55
Model uniform (C=1000.0): accuracy = 0.45
Model dirac (C=1000.0): accuracy = 0.60
Model uniform (C=0.0001): accuracy = 0.62
Model uniform (C=0.1): accuracy = 0.47
Model uniform (C=0.01): accuracy = 0.62
Model uniform (C=0.001): accuracy = 0.62
Model dirac (C=0.0001): accuracy = 0.62
Model multitask (C=1000.0): accuracy = 0.55
Model dirac (C=0.1): accuracy = 0.60
Model uniform (C=100.0): accuracy = 0.45
Model dirac (C=0.01): accuracy = 0.62
Model dirac (C=0.001): accuracy = 0.62
Model uniform (C=1.0): accuracy = 0.46
Model dirac (C=100.0): accuracy = 0.60
Model uniform (C=10.0): accuracy = 0.45
Model dirac (C=1.0): accuracy = 0.60
Model multitask (C=0.0001): accuracy = 0.62
Model multitask (C=0.1): accuracy = 0.54
```

```
Model multitask (C=0.01): accuracy = 0.62
Model dirac (C=10.0): accuracy = 0.60
Model multitask (C=0.001): accuracy = 0.62
Model multitask (C=100.0): accuracy = 0.55
Model multitask (C=1.0): accuracy = 0.55
Model multitask (C=10.0): accuracy = 0.55
Model uniform (C=1000.0): accuracy = 0.44
Model dirac (C=1000.0): accuracy = 0.47
Model uniform (C=0.0001): accuracy = 0.49
Model uniform (C=0.01): accuracy = 0.49
Model multitask (C=1000.0): accuracy = 0.60
Model uniform (C=0.001): accuracy = 0.49
Model dirac (C=0.0001): accuracy = 0.49
Model uniform (C=0.1): accuracy = 0.49
Model dirac (C=0.01): accuracy = 0.49
Model uniform (C=100.0): accuracy = 0.44
Model dirac (C=0.001): accuracy = 0.49
Model uniform (C=10.0): accuracy = 0.44
Model dirac (C=0.1): accuracy = 0.47
Model uniform (C=1.0): accuracy = 0.44
Model dirac (C=100.0): accuracy = 0.47
Model dirac (C=10.0): accuracy = 0.47
Model multitask (C=0.0001): accuracy = 0.49
Model dirac (C=1.0): accuracy = 0.47
Model multitask (C=0.01): accuracy = 0.49
Model multitask (C=0.1): accuracy = 0.45
Model multitask (C=0.001): accuracy = 0.49
Model multitask (C=100.0): accuracy = 0.60
Model multitask (C=10.0): accuracy = 0.60
Model multitask (C=1.0): accuracy = 0.60
Model uniform (C=1000.0): accuracy = 0.43
Model dirac (C=1000.0): accuracy = 0.50
Model multitask (C=1000.0): accuracy = 0.56
Model uniform (C=0.0001): accuracy = 0.52
Model uniform (C=0.1): accuracy = 0.47
Model uniform (C=0.01): accuracy = 0.52
Model uniform (C=0.001): accuracy = 0.52
Model dirac (C=0.0001): accuracy = 0.52
Model dirac (C=0.1): accuracy = 0.50
Model uniform (C=1.0): accuracy = 0.43
Model dirac (C=0.001): accuracy = 0.52
Model uniform (C=100.0): accuracy = 0.43
Model dirac (C=0.01): accuracy = 0.52
Model dirac (C=1.0): accuracy = 0.50
Model uniform (C=10.0): accuracy = 0.43
Model dirac (C=100.0): accuracy = 0.50
Model multitask (C=0.0001): accuracy = 0.52
Model multitask (C=0.1): accuracy = 0.53
```

```
Model multitask (C=0.01): accuracy = 0.52
Model multitask (C=0.001): accuracy = 0.52
Model dirac (C=10.0): accuracy = 0.50
Model multitask (C=1.0): accuracy = 0.56
Model multitask (C=100.0): accuracy = 0.56
Model multitask (C=10.0): accuracy = 0.56
Model uniform (C=10000.0): accuracy = 0.43
Model dirac (C=10000.0): accuracy = 0.54
Model multitask (C=10000.0): accuracy = 0.56
Model uniform (C=10000.0): accuracy = 0.40
Model dirac (C=10000.0): accuracy = 0.60
Model multitask (C=10000.0): accuracy = 0.53
Model uniform (C=10000.0): accuracy = 0.48
Model dirac (C=10000.0): accuracy = 0.52
Model multitask (C=10000.0): accuracy = 0.55
Model uniform (C=10000.0): accuracy = 0.44
Model dirac (C=10000.0): accuracy = 0.51
Model multitask (C=10000.0): accuracy = 0.51
Model uniform (C=10000.0): accuracy = 0.40
Model dirac (C=10000.0): accuracy = 0.58
Model multitask (C=10000.0): accuracy = 0.52
Model uniform (C=10000.0): accuracy = 0.47
Model dirac (C=10000.0): accuracy = 0.50
Model multitask (C=10000.0): accuracy = 0.56
Model uniform (C=10000.0): accuracy = 0.42
Model dirac (C=10000.0): accuracy = 0.50
Model multitask (C=10000.0): accuracy = 0.54
Model uniform (C=10000.0): accuracy = 0.43
Model dirac (C=10000.0): accuracy = 0.54
Model multitask (C=10000.0): accuracy = 0.57
Model uniform (C=10000.0): accuracy = 0.43
Model dirac (C=10000.0): accuracy = 0.47
Model multitask (C=10000.0): accuracy = 0.55
Model uniform (C=10000.0): accuracy = 0.46
Model dirac (C=10000.0): accuracy = 0.53
Model multitask (C=10000.0): accuracy = 0.60
```

The values above are for the individual cross validation steps, so they appear 10 times each. We will print the averaged values next:

[11]:
```
output = []
for C, model in scores.items():
    for name, results in model.items():
        output.append(f"Model {name} (C={C}): accuracy = {results['accuracy']:2.
 ↪2f}")
print("\n".join(sorted(output)))
```

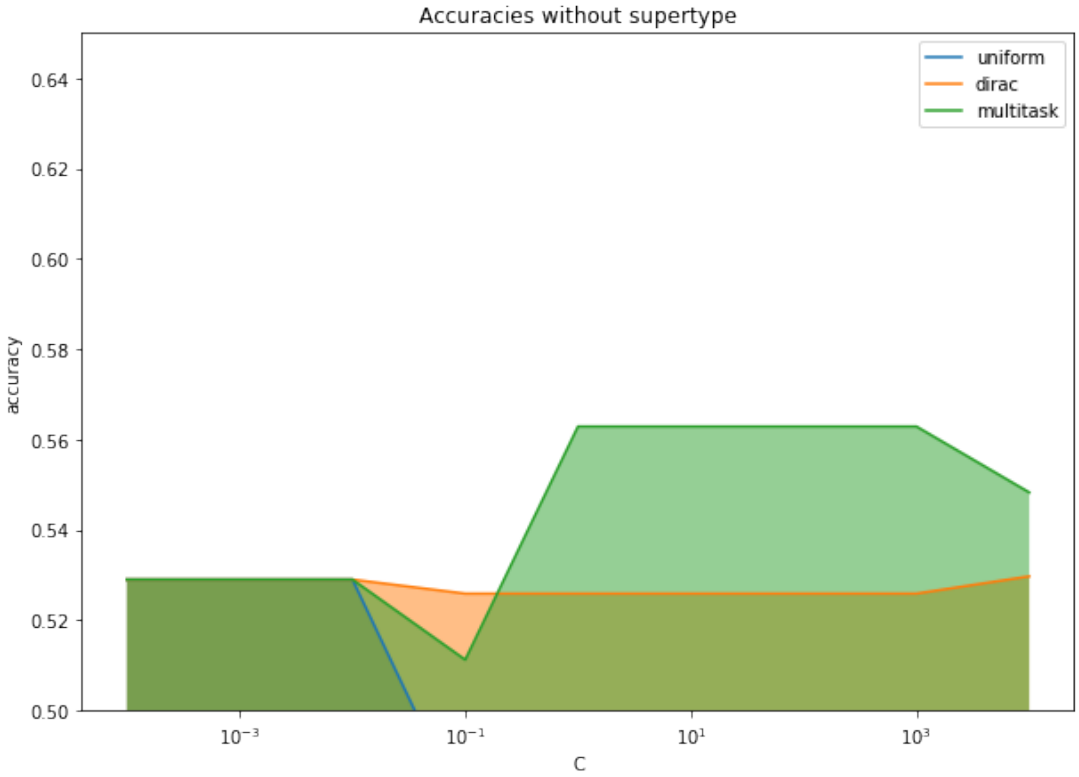
```
Model dirac (C=0.0001): accuracy = 0.53
```

```
Model dirac (C=0.001): accuracy = 0.53
Model dirac (C=0.01): accuracy = 0.53
Model dirac (C=0.1): accuracy = 0.53
Model dirac (C=1.0): accuracy = 0.53
Model dirac (C=10.0): accuracy = 0.53
Model dirac (C=100.0): accuracy = 0.53
Model dirac (C=1000.0): accuracy = 0.53
Model dirac (C=10000.0): accuracy = 0.53
Model multitask (C=0.0001): accuracy = 0.53
Model multitask (C=0.001): accuracy = 0.53
Model multitask (C=0.01): accuracy = 0.53
Model multitask (C=0.1): accuracy = 0.51
Model multitask (C=1.0): accuracy = 0.56
Model multitask (C=10.0): accuracy = 0.56
Model multitask (C=100.0): accuracy = 0.56
Model multitask (C=1000.0): accuracy = 0.56
Model multitask (C=10000.0): accuracy = 0.55
Model uniform (C=0.0001): accuracy = 0.53
Model uniform (C=0.001): accuracy = 0.53
Model uniform (C=0.01): accuracy = 0.53
Model uniform (C=0.1): accuracy = 0.48
Model uniform (C=1.0): accuracy = 0.44
Model uniform (C=10.0): accuracy = 0.44
Model uniform (C=100.0): accuracy = 0.44
Model uniform (C=1000.0): accuracy = 0.44
Model uniform (C=10000.0): accuracy = 0.43
```

[12]:
```python
def scores_to_accuracy_dict(scores):
    accuracies = {n: [] for n in scores[1].keys()}
    for C, model in sorted(scores.items(), key=lambda x: x[0]):
        for name, results in model.items():
            accuracies[name].append(results["accuracy"])
    return accuracies
```

[13]:
```python
def plot_accuracies(accuracies, Cs=Cs, size=(10, 7), ylim=(0.5, 0.65),␣
 ↪title=""):
    plt.figure(figsize=size)
    for name, accuracy in accuracies.items():
        plt.plot(Cs, accuracy, label=name)
        plt.fill_between(Cs, accuracy, alpha=0.5)
    plt.xlabel("C")
    plt.ylabel("accuracy")
    plt.xscale("log")
    plt.ylim(ylim)
    plt.title(title)
    plt.legend()
    plt.show()
```

[14]:
```
accuracies = scores_to_accuracy_dict(scores)
plot_accuracies(accuracies, title="Accuracies without supertype")
```



1.3 d1) ROC and AUC + e1) AUC and Accuracy

For better readability we will only generate ROC-curves for the fires CV-run. We could easily do more, by just looping over all ten results, but this way we would get $9 * 10 * 3 = 270$ plots, which is not really readable. One could also plot all CV runs in one plot, but that would also be confusing.

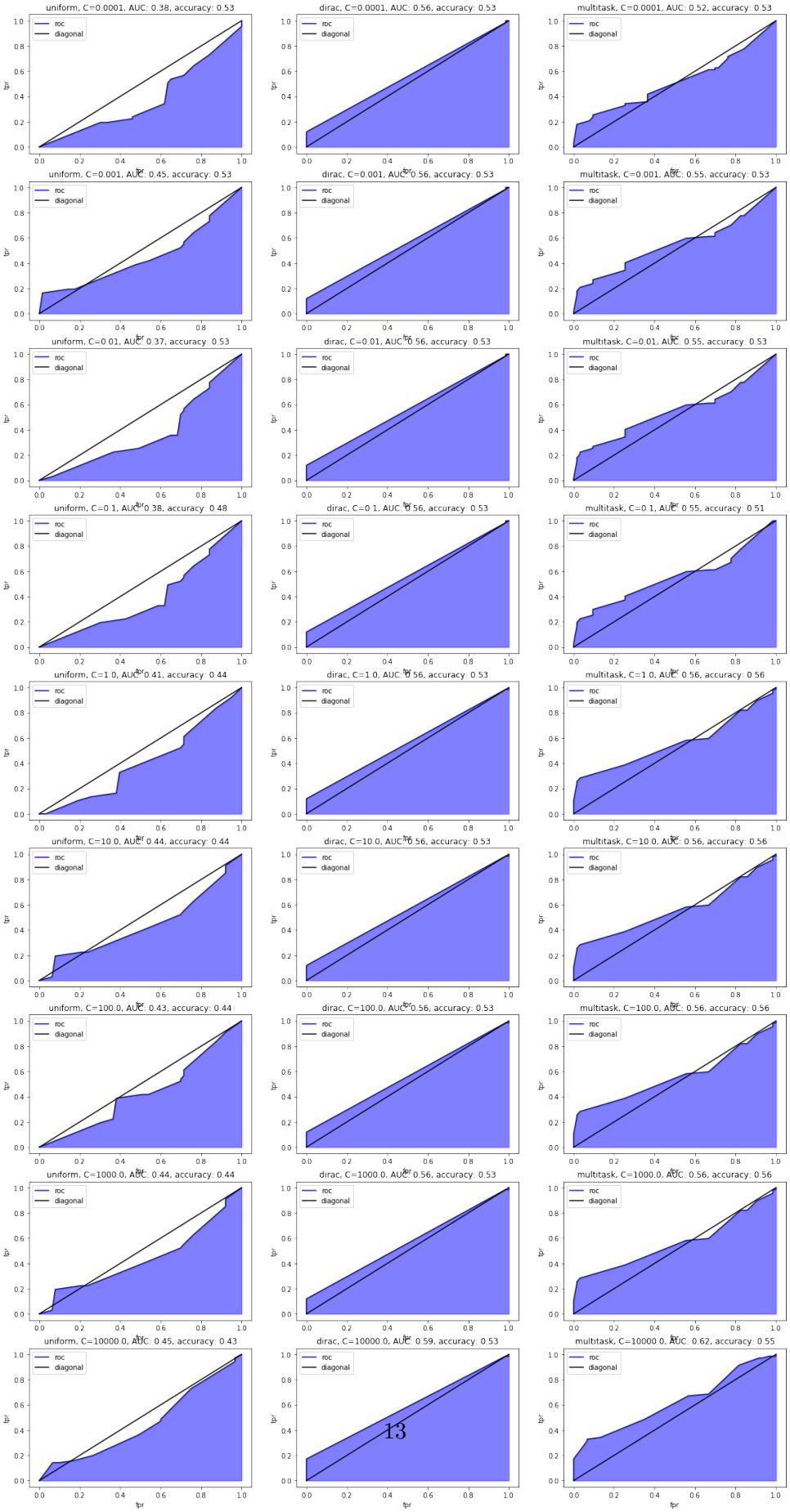
[18]:
```
def calc_roc(scores):
    out = {C: dict() for C in scores.keys()}
    for C, model in scores.items():
        for name, model_stats in model.items():
            result = model_stats["results"][0]
            truth = model_stats["truths"][0]
            fpr, tpr, t = roc_curve(truth, result)
            auc = roc_auc_score(truth, result)
            out[C][name] = {
                "tpr": tpr,
                "fpr": fpr,
                "auc": auc,
                "accuracy": model_stats["accuracy"]
```

```
            }
    return out
```

[23]:
```
def plot_roc(roc_data):
    fig, axss = plt.subplots(
        len(roc_data.keys()),
        len(roc_data[1].keys()),
        figsize=(20, 40)
    )
    for (C, model), axs in zip(roc_data.items(), axss):
        for (name, model_stats), ax in zip(model.items(), axs):

            ax.plot(model_stats["fpr"], model_stats["tpr"], label="roc",␣
↪color="blue")
            ax.fill_between(model_stats["fpr"], model_stats["tpr"],␣
↪color="blue", alpha=0.5)
            ax.plot([0,1], [0,1], label="diagonal", color="black")
            ax.legend()
            ax.set_xlabel("fpr")
            ax.set_ylabel("tpr")
            ax.set_title(f"{name}, C={C}, AUC: {model_stats['auc']:2.2f},␣
↪accuracy: {model_stats['accuracy']:2.2f}")
```

[24]:
```
roc_data = calc_roc(scores)
plot_roc(roc_data)
```


uniform, C=0.0001, AUC: 0.38, accuracy: 0.53
dirac, C=0.0001, AUC: 0.56, accuracy: 0.53
multitask, C=0.0001, AUC: 0.52, accuracy: 0.53
uniform, C=0.001, AUC: 0.45, accuracy: 0.53
dirac, C=0.001, AUC: 0.56, accuracy: 0.53
multitask, C=0.001, AUC: 0.55, accuracy: 0.53
uniform, C=0.01, AUC: 0.37, accuracy: 0.53
dirac, C=0.01, AUC: 0.56, accuracy: 0.53
multitask, C=0.01, AUC: 0.55, accuracy: 0.53
uniform, C=0.1, AUC: 0.38, accuracy: 0.48
dirac, C=0.1, AUC: 0.56, accuracy: 0.53
multitask, C=0.1, AUC: 0.55, accuracy: 0.51
uniform, C=1.0, AUC: 0.41, accuracy: 0.44
dirac, C=1.0, AUC: 0.56, accuracy: 0.53
multitask, C=1.0, AUC: 0.56, accuracy: 0.56
uniform, C=10.0, AUC: 0.44, accuracy: 0.44
dirac, C=10.0, AUC: 0.56, accuracy: 0.53
multitask, C=10.0, AUC: 0.56, accuracy: 0.56
uniform, C=100.0, AUC: 0.43, accuracy: 0.44
dirac, C=100.0, AUC: 0.56, accuracy: 0.53
multitask, C=100.0, AUC: 0.56, accuracy: 0.56
uniform, C=1000.0, AUC: 0.44, accuracy: 0.44
dirac, C=1000.0, AUC: 0.56, accuracy: 0.53
multitask, C=1000.0, AUC: 0.56, accuracy: 0.56
uniform, C=10000.0, AUC: 0.45, accuracy: 0.43
dirac, C=10000.0, AUC: 0.59, accuracy: 0.53
multitask, C=10000.0, AUC: 0.62, accuracy: 0.55
roc
diagonal
tpr
fpr

2 c) Add supertype data

[25]:
```
supertypes_data = pd.read_csv("supertype.csv")
```

[26]:
```
supertypes_data
```

[26]:
```
  Supertype                          Motif  \
0        A1          x[TI(SVLM)]xxxxxx[WFY]
1        A2        x[LIVMATQ]xxxxxx[LIVMAT]
2        A3            x[AILMVST]xxxxxx[RK]
3       A24  x[YF(WIVLMT)]xxxxxx[FI(YWLM)]
4        B7            x[P]xxxxxx[ALIMVFWY]
5       B27          x[RKH]xxxxxx[FLY(WMI)]
6       B44         x[E(D)]xxxxxx[FWYLIMVA]
7       B58          x[AST]xxxxxx[FWY(LIV)]
8       B62     x[QL(IVMP)]xxxxxx[FWY(MIV)]

                                      Genotypes
0  A*0101, A*0102, A*2501, A*2601, A*2604, A*3201…
1  A*0201, A*0202, A*0203, A*0204, A*0205, A*0206…
2             A*0301, A*1101, A*3101, A*3301, A*6801
3  A*2301, A*2402, A*2403, A*2404, A*3001, A*3002…
4  B*0702, B*0703, B*0704, B*0705, B*1508, B*3501…
5  B*1401, B*1402, B*1503, B*1509, B*1510, B*1518…
6  B*18, B*3701, B*4001, B*4006, B*4101, B*4402, …
7               B*1516, B*1517, B*5701, B*5702, B*58
8  B*1301, B*1302, B*1501, B*1502, B*1506, B*1512…
```

[27]:
```
allele_supertype_mapping = []
for i, row in supertypes_data.iterrows():
    s = row["Supertype"]
    for allele in row["Genotypes"].split(", "):
        a = f"HLA-{allele[:-2]}:{allele[-2:]}"
        allele_supertype_mapping.append((a, s))
```

[28]:
```
supertype_df = pd.DataFrame(allele_supertype_mapping, columns=["allele",␣
↪"supertype"])
supertype_df.head()
```

[28]:
```
        allele supertype
0  HLA-A*01:01        A1
1  HLA-A*01:02        A1
2  HLA-A*25:01        A1
```

```
3  HLA-A*26:01         A1
4  HLA-A*26:04         A1
```

[29]:
```
combined_data = binding_data.join(supertype_df.set_index("allele"), on="␣
↪allele")
combined_data.head()
```

[29]:
```
   Epitope ID  binding information       allele  peptide sequence  \
0       52886             Positive  HLA-A*01:01         QYDPVAALF
1       71650             Positive  HLA-A*01:01         VVEKQSGLY
2      150319             Positive  HLA-A*01:01         ITEAELTGY
3      141221             Positive  HLA-A*01:01         ATDSLNNEY
4       24819             Positive  HLA-A*01:01         HSNLNDATY

    binding label supertype
0               1        A1
1               1        A1
2               1        A1
3               1        A1
4               1        A1
```

[30]:
```
spa_data = combined_data[["supertype", " peptide sequence", " allele"]].
↪to_numpy()
label_data = combined_data[" binding label"].to_numpy()
```

[31]:
```
def do_cv2(C):
    kf = KFold(n_splits=10, shuffle=True)

    scores = dict()
    names = ["uniform", "dirac", "multitask"]
    for name in names:
        scores[name] = {
            "accuracy": [],
            "results": [],
            "truths": [],
            "classifiers": []
        }

    for train_indices, test_indices in kf.split(pa_data):



        Xtrain = spa_data[train_indices]
        Ytrain = label_data[train_indices]
        Xtest = spa_data[test_indices]
        Ytest = label_data[test_indices]
```

```
        K_train_s = kernel(k_dirac)(Xtrain[:, 0], Xtrain[:, 0])
        K_test_s = kernel(k_dirac)(Xtest[:, 0], Xtrain[:, 0])
        K_train_p = kernel(k_linseq)(Xtrain[:, 1], Xtrain[:, 1])
        K_test_p = kernel(k_linseq)(Xtest[:, 1], Xtrain[:, 1])

        for n, k in zip(
                        names,
                        [k_uniform, k_dirac, k_multitask]
        ):
            e = svm.SVC(
                kernel="precomputed",
                C=C
            )
            K_train_a = kernel(k)(Xtrain[:, 2], Xtrain[:, 2])
            K_test_a = kernel(k)(Xtest[:, 2], Xtrain[:, 2])

            Ktrain = join_kernel_matricies(K_train_s, K_train_p, K_train_a)
            Ktest = join_kernel_matricies(K_test_s, K_test_p, K_test_a)


            e.fit(Ktrain, Ytrain)
            score = e.score(Ktest, Ytest)
            scores[n]["accuracy"].append(score)
            print(f"Model {n} (C={C}): accuracy = {score:2.2f}")

            results = e.decision_function(Ktest)
            scores[n]["results"].append(results)
            scores[n]["truths"].append(Ytest)

            scores[n]["classifiers"].append(e)

    for n in scores.keys():
        scores[n]["accuracy"] = np.array(scores[n]["accuracy"]).mean()

    return scores
```

[32]:
```
futures = []
with ProcessPoolExecutor() as ppe:
    for C in Cs:
        futures.append((C, ppe.submit(do_cv2, C)))
scores = dict()
for C, future in futures:
    scores[C] = future.result()
```

```
Model uniform (C=0.01): accuracy = 0.52
Model uniform (C=0.001): accuracy = 0.52
```

```
Model uniform (C=0.1): accuracy = 0.63
Model uniform (C=1.0): accuracy = 0.64
Model uniform (C=0.0001): accuracy = 0.52
Model uniform (C=10.0): accuracy = 0.62
Model dirac (C=0.01): accuracy = 0.52
Model dirac (C=0.001): accuracy = 0.52
Model dirac (C=0.1): accuracy = 0.64
Model dirac (C=1.0): accuracy = 0.63
Model dirac (C=0.0001): accuracy = 0.52
Model uniform (C=1000.0): accuracy = 0.57
Model uniform (C=100.0): accuracy = 0.57
Model dirac (C=10.0): accuracy = 0.58
Model multitask (C=0.01): accuracy = 0.52
Model multitask (C=0.001): accuracy = 0.52
Model dirac (C=1000.0): accuracy = 0.54
Model multitask (C=0.1): accuracy = 0.61
Model multitask (C=0.0001): accuracy = 0.52
Model dirac (C=100.0): accuracy = 0.54
Model multitask (C=1.0): accuracy = 0.65
Model multitask (C=10.0): accuracy = 0.60
Model multitask (C=1000.0): accuracy = 0.55
Model multitask (C=100.0): accuracy = 0.55
Model uniform (C=0.01): accuracy = 0.51
Model uniform (C=0.0001): accuracy = 0.51
Model uniform (C=0.001): accuracy = 0.51
Model dirac (C=0.01): accuracy = 0.51
Model uniform (C=0.1): accuracy = 0.62
Model dirac (C=0.0001): accuracy = 0.51
Model uniform (C=1.0): accuracy = 0.63
Model dirac (C=0.001): accuracy = 0.51
Model dirac (C=0.1): accuracy = 0.60
Model uniform (C=10.0): accuracy = 0.65
Model dirac (C=1.0): accuracy = 0.62
Model multitask (C=0.01): accuracy = 0.51
Model multitask (C=0.0001): accuracy = 0.51
Model multitask (C=0.001): accuracy = 0.51
Model dirac (C=10.0): accuracy = 0.60
Model multitask (C=0.1): accuracy = 0.62
Model uniform (C=1000.0): accuracy = 0.65
Model multitask (C=1.0): accuracy = 0.64
Model uniform (C=100.0): accuracy = 0.65
Model dirac (C=1000.0): accuracy = 0.60
Model dirac (C=100.0): accuracy = 0.60
Model multitask (C=10.0): accuracy = 0.60
Model multitask (C=1000.0): accuracy = 0.60
Model multitask (C=100.0): accuracy = 0.60
Model uniform (C=0.0001): accuracy = 0.55
Model uniform (C=0.01): accuracy = 0.55
```

```
Model uniform (C=0.1): accuracy = 0.60
Model dirac (C=0.0001): accuracy = 0.55
Model dirac (C=0.01): accuracy = 0.55
Model uniform (C=0.001): accuracy = 0.55
Model uniform (C=1.0): accuracy = 0.61
Model dirac (C=0.1): accuracy = 0.60
Model dirac (C=0.001): accuracy = 0.55
Model dirac (C=1.0): accuracy = 0.56
Model multitask (C=0.0001): accuracy = 0.55
Model multitask (C=0.01): accuracy = 0.55
Model uniform (C=10.0): accuracy = 0.65
Model multitask (C=0.1): accuracy = 0.62
Model multitask (C=0.001): accuracy = 0.55
Model multitask (C=1.0): accuracy = 0.59
Model dirac (C=10.0): accuracy = 0.52
Model uniform (C=1000.0): accuracy = 0.64
Model uniform (C=100.0): accuracy = 0.64
Model dirac (C=1000.0): accuracy = 0.52
Model multitask (C=10.0): accuracy = 0.55
Model dirac (C=100.0): accuracy = 0.52
Model multitask (C=1000.0): accuracy = 0.55
Model multitask (C=100.0): accuracy = 0.55
Model uniform (C=0.0001): accuracy = 0.52
Model dirac (C=0.0001): accuracy = 0.52
Model uniform (C=0.01): accuracy = 0.52
Model uniform (C=0.1): accuracy = 0.68
Model dirac (C=0.01): accuracy = 0.52
Model uniform (C=0.001): accuracy = 0.52
Model dirac (C=0.1): accuracy = 0.63
Model multitask (C=0.0001): accuracy = 0.52
Model uniform (C=1.0): accuracy = 0.58
Model dirac (C=0.001): accuracy = 0.52
Model multitask (C=0.01): accuracy = 0.52
Model dirac (C=1.0): accuracy = 0.57
Model multitask (C=0.1): accuracy = 0.67
Model multitask (C=0.001): accuracy = 0.52
Model uniform (C=10.0): accuracy = 0.62
Model multitask (C=1.0): accuracy = 0.58
Model dirac (C=10.0): accuracy = 0.56
Model uniform (C=1000.0): accuracy = 0.62
Model multitask (C=10.0): accuracy = 0.58
Model uniform (C=100.0): accuracy = 0.62
Model dirac (C=1000.0): accuracy = 0.57
Model dirac (C=100.0): accuracy = 0.57
Model multitask (C=1000.0): accuracy = 0.57
Model multitask (C=100.0): accuracy = 0.57
Model uniform (C=0.0001): accuracy = 0.52
Model uniform (C=0.1): accuracy = 0.64
```

```
Model dirac (C=0.0001): accuracy = 0.52
Model uniform (C=0.01): accuracy = 0.52
Model dirac (C=0.1): accuracy = 0.62
Model dirac (C=0.01): accuracy = 0.52
Model uniform (C=0.001): accuracy = 0.52
Model multitask (C=0.0001): accuracy = 0.52
Model dirac (C=0.001): accuracy = 0.52
Model uniform (C=1.0): accuracy = 0.58
Model multitask (C=0.1): accuracy = 0.64
Model multitask (C=0.01): accuracy = 0.52
Model dirac (C=1.0): accuracy = 0.59
Model multitask (C=0.001): accuracy = 0.52
Model uniform (C=10.0): accuracy = 0.63
Model multitask (C=1.0): accuracy = 0.62
Model dirac (C=10.0): accuracy = 0.61
Model multitask (C=10.0): accuracy = 0.62
Model uniform (C=100.0): accuracy = 0.60
Model uniform (C=1000.0): accuracy = 0.62
Model dirac (C=100.0): accuracy = 0.61
Model dirac (C=1000.0): accuracy = 0.61
Model multitask (C=100.0): accuracy = 0.62
Model multitask (C=1000.0): accuracy = 0.62
Model uniform (C=0.0001): accuracy = 0.50
Model uniform (C=0.1): accuracy = 0.63
Model dirac (C=0.0001): accuracy = 0.50
Model dirac (C=0.1): accuracy = 0.59
Model uniform (C=0.01): accuracy = 0.50
Model dirac (C=0.01): accuracy = 0.50
Model uniform (C=0.001): accuracy = 0.50
Model multitask (C=0.0001): accuracy = 0.50
Model multitask (C=0.1): accuracy = 0.58
Model dirac (C=0.001): accuracy = 0.50
Model uniform (C=1.0): accuracy = 0.58
Model multitask (C=0.01): accuracy = 0.50
Model dirac (C=1.0): accuracy = 0.52
Model multitask (C=0.001): accuracy = 0.50
Model multitask (C=1.0): accuracy = 0.53
Model uniform (C=10.0): accuracy = 0.56
Model dirac (C=10.0): accuracy = 0.55
Model multitask (C=10.0): accuracy = 0.57
Model uniform (C=100.0): accuracy = 0.57
Model uniform (C=1000.0): accuracy = 0.57
Model dirac (C=100.0): accuracy = 0.57
Model dirac (C=1000.0): accuracy = 0.57
Model multitask (C=100.0): accuracy = 0.58
Model multitask (C=1000.0): accuracy = 0.58
Model uniform (C=0.0001): accuracy = 0.55
Model uniform (C=0.1): accuracy = 0.61
```

```
Model dirac (C=0.0001): accuracy = 0.55
Model dirac (C=0.1): accuracy = 0.61
Model uniform (C=0.01): accuracy = 0.55
Model dirac (C=0.01): accuracy = 0.55
Model uniform (C=0.001): accuracy = 0.55
Model multitask (C=0.0001): accuracy = 0.55
Model multitask (C=0.1): accuracy = 0.61
Model dirac (C=0.001): accuracy = 0.55
Model multitask (C=0.01): accuracy = 0.55
Model uniform (C=1.0): accuracy = 0.62
Model dirac (C=1.0): accuracy = 0.60
Model multitask (C=0.001): accuracy = 0.55
Model multitask (C=1.0): accuracy = 0.62
Model uniform (C=10.0): accuracy = 0.65
Model dirac (C=10.0): accuracy = 0.62
Model multitask (C=10.0): accuracy = 0.60
Model uniform (C=100.0): accuracy = 0.65
Model uniform (C=1000.0): accuracy = 0.64
Model dirac (C=100.0): accuracy = 0.61
Model dirac (C=1000.0): accuracy = 0.59
Model multitask (C=100.0): accuracy = 0.59
Model multitask (C=1000.0): accuracy = 0.58
Model uniform (C=0.0001): accuracy = 0.62
Model uniform (C=0.1): accuracy = 0.64
Model dirac (C=0.0001): accuracy = 0.62
Model dirac (C=0.1): accuracy = 0.63
Model uniform (C=0.01): accuracy = 0.62
Model multitask (C=0.0001): accuracy = 0.62
Model dirac (C=0.01): accuracy = 0.62
Model multitask (C=0.1): accuracy = 0.63
Model uniform (C=0.001): accuracy = 0.62
Model dirac (C=0.001): accuracy = 0.62
Model multitask (C=0.01): accuracy = 0.62
Model uniform (C=1.0): accuracy = 0.63
Model multitask (C=0.001): accuracy = 0.62
Model dirac (C=1.0): accuracy = 0.62
Model multitask (C=1.0): accuracy = 0.62
Model uniform (C=10.0): accuracy = 0.59
Model dirac (C=10.0): accuracy = 0.57
Model multitask (C=10.0): accuracy = 0.60
Model uniform (C=1000.0): accuracy = 0.63
Model uniform (C=100.0): accuracy = 0.63
Model dirac (C=1000.0): accuracy = 0.60
Model dirac (C=100.0): accuracy = 0.60
Model multitask (C=1000.0): accuracy = 0.62
Model multitask (C=100.0): accuracy = 0.62
Model uniform (C=0.0001): accuracy = 0.49
Model uniform (C=0.1): accuracy = 0.61
```

```
Model dirac (C=0.0001): accuracy = 0.49
Model dirac (C=0.1): accuracy = 0.60
Model uniform (C=0.01): accuracy = 0.49
Model multitask (C=0.0001): accuracy = 0.49
Model dirac (C=0.01): accuracy = 0.49
Model multitask (C=0.1): accuracy = 0.61
Model uniform (C=0.001): accuracy = 0.49
Model dirac (C=0.001): accuracy = 0.49
Model multitask (C=0.01): accuracy = 0.49
Model multitask (C=0.001): accuracy = 0.49
Model uniform (C=1.0): accuracy = 0.60
Model dirac (C=1.0): accuracy = 0.57
Model multitask (C=1.0): accuracy = 0.57
Model uniform (C=10.0): accuracy = 0.53
Model dirac (C=10.0): accuracy = 0.53
Model multitask (C=10.0): accuracy = 0.54
Model uniform (C=100.0): accuracy = 0.51
Model uniform (C=1000.0): accuracy = 0.51
Model dirac (C=100.0): accuracy = 0.53
Model dirac (C=1000.0): accuracy = 0.53
Model multitask (C=100.0): accuracy = 0.55
Model multitask (C=1000.0): accuracy = 0.55
Model uniform (C=0.0001): accuracy = 0.52
Model uniform (C=0.1): accuracy = 0.56
Model dirac (C=0.0001): accuracy = 0.52
Model dirac (C=0.1): accuracy = 0.56
Model uniform (C=0.01): accuracy = 0.52
Model dirac (C=0.01): accuracy = 0.52
Model multitask (C=0.0001): accuracy = 0.52
Model multitask (C=0.1): accuracy = 0.56
Model uniform (C=0.001): accuracy = 0.52
Model dirac (C=0.001): accuracy = 0.52
Model multitask (C=0.01): accuracy = 0.52
Model multitask (C=0.001): accuracy = 0.52
Model uniform (C=1.0): accuracy = 0.50
Model dirac (C=1.0): accuracy = 0.54
Model multitask (C=1.0): accuracy = 0.55
Model uniform (C=10.0): accuracy = 0.53
Model dirac (C=10.0): accuracy = 0.53
Model multitask (C=10.0): accuracy = 0.54
Model uniform (C=100.0): accuracy = 0.50
Model uniform (C=1000.0): accuracy = 0.50
Model dirac (C=100.0): accuracy = 0.53
Model dirac (C=1000.0): accuracy = 0.53
Model multitask (C=100.0): accuracy = 0.54
Model multitask (C=1000.0): accuracy = 0.54
Model uniform (C=10000.0): accuracy = 0.63
Model dirac (C=10000.0): accuracy = 0.61
```

```
Model multitask (C=10000.0): accuracy = 0.59
Model uniform (C=10000.0): accuracy = 0.52
Model dirac (C=10000.0): accuracy = 0.54
Model multitask (C=10000.0): accuracy = 0.52
Model uniform (C=10000.0): accuracy = 0.64
Model dirac (C=10000.0): accuracy = 0.62
Model multitask (C=10000.0): accuracy = 0.64
Model uniform (C=10000.0): accuracy = 0.55
Model dirac (C=10000.0): accuracy = 0.52
Model multitask (C=10000.0): accuracy = 0.54
Model uniform (C=10000.0): accuracy = 0.60
Model dirac (C=10000.0): accuracy = 0.58
Model multitask (C=10000.0): accuracy = 0.62
Model uniform (C=10000.0): accuracy = 0.52
Model dirac (C=10000.0): accuracy = 0.57
Model multitask (C=10000.0): accuracy = 0.57
Model uniform (C=10000.0): accuracy = 0.60
Model dirac (C=10000.0): accuracy = 0.57
Model multitask (C=10000.0): accuracy = 0.58
Model uniform (C=10000.0): accuracy = 0.64
Model dirac (C=10000.0): accuracy = 0.60
Model multitask (C=10000.0): accuracy = 0.60
Model uniform (C=10000.0): accuracy = 0.60
Model dirac (C=10000.0): accuracy = 0.54
Model multitask (C=10000.0): accuracy = 0.58
Model uniform (C=10000.0): accuracy = 0.62
Model dirac (C=10000.0): accuracy = 0.61
Model multitask (C=10000.0): accuracy = 0.60
```

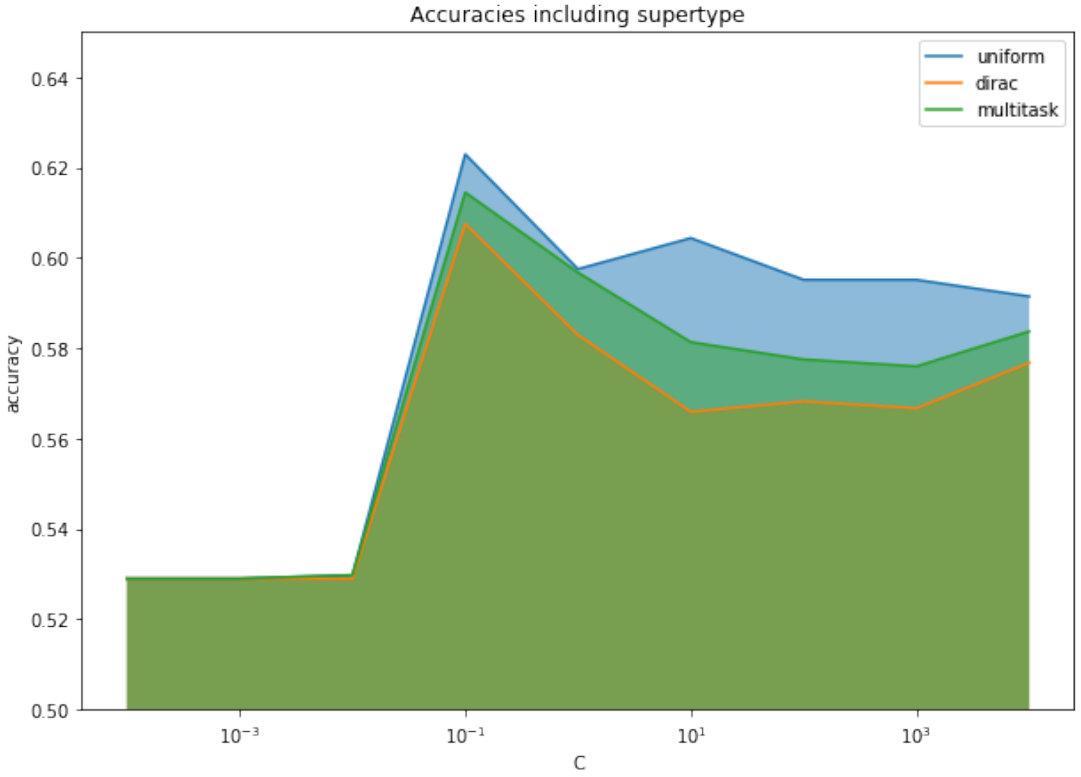
The values above are for the individual cross validation steps, so they appear 10 times each. We will print the averaged values next:

[33]:
```
output = []
for C, model in scores.items():
    for name, results in model.items():
        output.append(f"Model {name} (C={C}): accuracy = {results['accuracy']:2.
 ↪2f}")
print("\n".join(sorted(output)))
```

```
Model dirac (C=0.0001): accuracy = 0.53
Model dirac (C=0.001): accuracy = 0.53
Model dirac (C=0.01): accuracy = 0.53
Model dirac (C=0.1): accuracy = 0.61
Model dirac (C=1.0): accuracy = 0.58
Model dirac (C=10.0): accuracy = 0.57
Model dirac (C=100.0): accuracy = 0.57
Model dirac (C=1000.0): accuracy = 0.57
Model dirac (C=10000.0): accuracy = 0.58
```

```
Model multitask (C=0.0001): accuracy = 0.53
Model multitask (C=0.001): accuracy = 0.53
Model multitask (C=0.01): accuracy = 0.53
Model multitask (C=0.1): accuracy = 0.61
Model multitask (C=1.0): accuracy = 0.60
Model multitask (C=10.0): accuracy = 0.58
Model multitask (C=100.0): accuracy = 0.58
Model multitask (C=1000.0): accuracy = 0.58
Model multitask (C=10000.0): accuracy = 0.58
Model uniform (C=0.0001): accuracy = 0.53
Model uniform (C=0.001): accuracy = 0.53
Model uniform (C=0.01): accuracy = 0.53
Model uniform (C=0.1): accuracy = 0.62
Model uniform (C=1.0): accuracy = 0.60
Model uniform (C=10.0): accuracy = 0.60
Model uniform (C=100.0): accuracy = 0.60
Model uniform (C=1000.0): accuracy = 0.60
Model uniform (C=10000.0): accuracy = 0.59
```

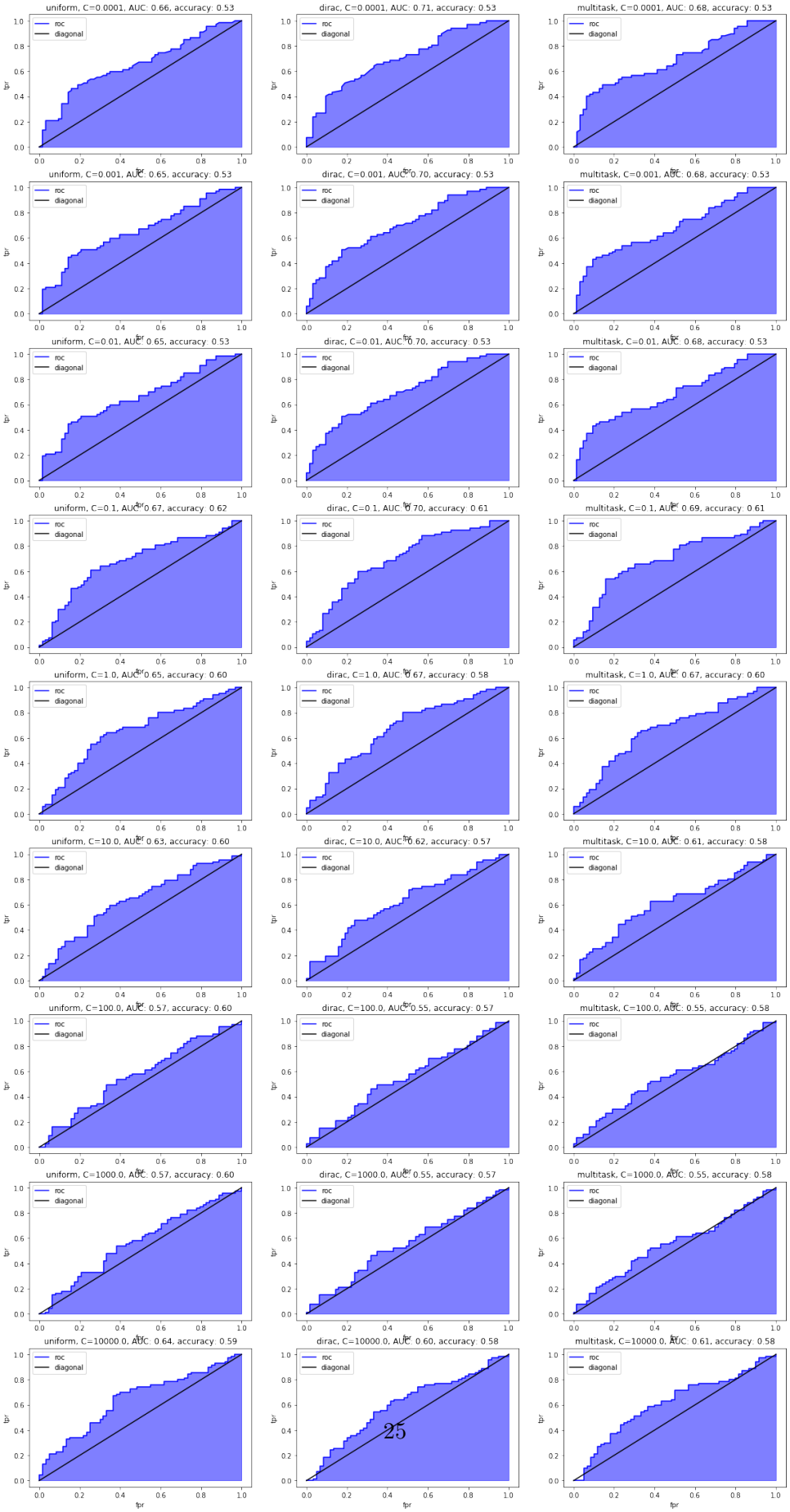
[34]:
```
accuracies = scores_to_accuracy_dict(scores)
plot_accuracies(accuracies, title="Accuracies including supertype")
```



Already from the accuracies on can see that the supertypes really help. Starting from $C = 10^{-1}$ all the models get reasonable accuracies (not great, but significantly better than random). Previously this was not the case and some accuracies even fell below 0.5. Looking at the ROC curves (below) gives a very similar picture: With the supertypes the curves are actually better than random, whereas before they looked pretty much like random performance.

2.1 d2) ROC and AUC + e2) AUC and Accuracy

[35]:
```
roc_data = calc_roc(scores)
plot_roc(roc_data)
```


uniform, C=0.0001, AUC: 0.66, accuracy: 0.53
dirac, C=0.0001, AUC: 0.71, accuracy: 0.53
multitask, C=0.0001, AUC: 0.68, accuracy: 0.53
uniform, C=0.001, AUC: 0.65, accuracy: 0.53
dirac, C=0.001, AUC: 0.70, accuracy: 0.53
multitask, C=0.001, AUC: 0.68, accuracy: 0.53
uniform, C=0.01, AUC: 0.65, accuracy: 0.53
dirac, C=0.01, AUC: 0.70, accuracy: 0.53
multitask, C=0.01, AUC: 0.68, accuracy: 0.53
uniform, C=0.1, AUC: 0.67, accuracy: 0.62
dirac, C=0.1, AUC: 0.70, accuracy: 0.61
multitask, C=0.1, AUC: 0.69, accuracy: 0.61
uniform, C=1.0, AUC: 0.65, accuracy: 0.60
dirac, C=1.0, AUC: 0.67, accuracy: 0.58
multitask, C=1.0, AUC: 0.67, accuracy: 0.60
uniform, C=10.0, AUC: 0.63, accuracy: 0.60
dirac, C=10.0, AUC: 0.62, accuracy: 0.57
multitask, C=10.0, AUC: 0.61, accuracy: 0.58
uniform, C=100.0, AUC: 0.57, accuracy: 0.60
dirac, C=100.0, AUC: 0.55, accuracy: 0.57
multitask, C=100.0, AUC: 0.55, accuracy: 0.58
uniform, C=1000.0, AUC: 0.57, accuracy: 0.60
dirac, C=1000.0, AUC: 0.55, accuracy: 0.57
multitask, C=1000.0, AUC: 0.55, accuracy: 0.58
uniform, C=10000.0, AUC: 0.64, accuracy: 0.59
dirac, C=10000.0, AUC: 0.60, accuracy: 0.58
multitask, C=10000.0, AUC: 0.61, accuracy: 0.58
roc
diagonal
tpr
fpr

2.2 d)

First up one can clearly say, that including the supertypes really helped. Both accuracies and AUC values increased after adding the supertype. Looking at the different kernels and values of C, there is not as much difference. For small C there is very little difference between the kernels, which is to be expected, since the model is strongly regularized. In general the best C seems to be about 0.1 where all three kernels do a reasonably good job. The best model is dirac kernel, $C = 0.1$ with an AUC of 0.7 and an accuracy of 0.61. This is not really clear cut, though since the model next to it (uniform, $C = 0.1$) achives a slightly higher accuracy (0.62), but looses on the AUC (0.67).

2.3 e)

One can see, that the accuracies and AUC values are heavily correlated. In general higher accuracy also means higher AUC. This is not always the case though. For example for iniform kernel and $C = 1000$ we get accuracy 0.6 and AUC 0.57. However for multitask $C = 10000$ we get accuracy 0.58 and AUC 0.61. This clearly shows, that accuracy can improve, while AUC get worse.

This is because accuracy and AUC measure different quantities. Accuracy measures how many predictions were correct, and does not compare in what way they are wrong if they are wrong. AUC measures the true positive rate on can get while tolerating a certain false positive rate. This is achived by altering the constant threshold added to the result of the SVM. This threshold is however not added for the computation of accuracy and therefore these metrics are fundamentally different.

Both, however, have the property that they can range from 0 to 1 where 1 is perfect, 0.5 is as good as random and anything less than that should really not happen. Therefore they are correlated, because a good model should have high accuracy and AUC. Therefore seeing one of these values gives you and idea about the quality of the model, as well as about the other metric, because they both depend on the model being good.