a02-lr-solution

November 2, 2024

0.0.1 Elizaveta Nosova (enosova), Artem Bisliouk (abisliou)

[105]:
```
import matplotlib.pyplot as plt
import numpy as np
import numpy.random
import numpy.linalg
import scipy.io
import scipy.stats
import sklearn.metrics

# setup plotting
from IPython import get_ipython
import psutil
inTerminal = not "IPKernelApp" in get_ipython().config
inJupyterNb = any(filter(lambda x: x.endswith("jupyter-notebook"), psutil.
  ↪Process().parent().cmdline()))
get_ipython().run_line_magic("matplotlib", "" if inTerminal else "notebook" if␣
  ↪inJupyterNb else "widget")
def nextplot():
    if inTerminal:
        plt.clf()     # this clears the current plot
    else:
        plt.figure()  # this creates a new plot
```

1 Load the data

[106]:
```
data = scipy.io.loadmat("data/spamData.mat")
X = data["Xtrain"]
N = X.shape[0]
D = X.shape[1]
Xtest = data["Xtest"]
Ntest = Xtest.shape[0]
y = data["ytrain"].squeeze().astype(int)
ytest = data["ytest"].squeeze().astype(int)

features = np.array(
    [
```

```
        "word_freq_make",
        "word_freq_address",
        "word_freq_all",
        "word_freq_3d",
        "word_freq_our",
        "word_freq_over",
        "word_freq_remove",
        "word_freq_internet",
        "word_freq_order",
        "word_freq_mail",
        "word_freq_receive",
        "word_freq_will",
        "word_freq_people",
        "word_freq_report",
        "word_freq_addresses",
        "word_freq_free",
        "word_freq_business",
        "word_freq_email",
        "word_freq_you",
        "word_freq_credit",
        "word_freq_your",
        "word_freq_font",
        "word_freq_000",
        "word_freq_money",
        "word_freq_hp",
        "word_freq_hpl",
        "word_freq_george",
        "word_freq_650",
        "word_freq_lab",
        "word_freq_labs",
        "word_freq_telnet",
        "word_freq_857",
        "word_freq_data",
        "word_freq_415",
        "word_freq_85",
        "word_freq_technology",
        "word_freq_1999",
        "word_freq_parts",
        "word_freq_pm",
        "word_freq_direct",
        "word_freq_cs",
        "word_freq_meeting",
        "word_freq_original",
        "word_freq_project",
        "word_freq_re",
        "word_freq_edu",
        "word_freq_table",
```

```
        "word_freq_conference",
        "char_freq_;",
        "char_freq_(",
        "char_freq_[",
        "char_freq_!",
        "char_freq_$",
        "char_freq_#",
        "capital_run_length_average",
        "capital_run_length_longest",
        "capital_run_length_total",
    ]
)
```

2 1. Dataset Statistics

[107]:
```
# look some dataset statistics
scipy.stats.describe(X)
```

[107]:
```
DescribeResult(nobs=3065, minmax=(array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 1., 1.]), array([4.5400e+00, 1.4280e+01, 5.1000e+00,
4.2810e+01, 9.0900e+00,
       3.5700e+00, 7.2700e+00, 1.1110e+01, 3.3300e+00, 1.8180e+01,
       2.0000e+00, 9.6700e+00, 5.5500e+00, 5.5500e+00, 2.8600e+00,
       1.0160e+01, 7.1400e+00, 9.0900e+00, 1.8750e+01, 6.3200e+00,
       1.1110e+01, 1.7100e+01, 5.4500e+00, 9.0900e+00, 2.0000e+01,
       1.4280e+01, 3.3330e+01, 4.7600e+00, 1.4280e+01, 4.7600e+00,
       4.7600e+00, 4.7600e+00, 1.8180e+01, 4.7600e+00, 2.0000e+01,
       7.6900e+00, 6.8900e+00, 7.4000e+00, 9.7500e+00, 4.7600e+00,
       7.1400e+00, 1.4280e+01, 3.5700e+00, 2.0000e+01, 2.1420e+01,
       1.6700e+01, 2.1200e+00, 1.0000e+01, 4.3850e+00, 9.7520e+00,
       4.0810e+00, 3.2478e+01, 6.0030e+00, 1.9829e+01, 1.1025e+03,
       9.9890e+03, 1.5841e+04])), mean=array([1.10818923e-01, 2.28486134e-01,
2.74153344e-01, 6.29690049e-02,
       3.17787928e-01, 9.57553018e-02, 1.13546493e-01, 1.07216966e-01,
       8.89233279e-02, 2.41719413e-01, 5.81305057e-02, 5.37432300e-01,
       9.26231648e-02, 4.96639478e-02, 5.07210440e-02, 2.35334421e-01,
       1.47197390e-01, 1.86600326e-01, 1.66121044e+00, 7.63066884e-02,
       8.19592170e-01, 1.22727569e-01, 1.02006525e-01, 8.90799347e-02,
       5.29800979e-01, 2.62071778e-01, 7.71507341e-01, 1.14323002e-01,
       1.09487765e-01, 9.92952692e-02, 6.28156607e-02, 4.90342577e-02,
       9.27471452e-02, 4.96019576e-02, 1.02156607e-01, 9.93050571e-02,
       1.43285481e-01, 1.24274062e-02, 7.55921697e-02, 6.60456770e-02,
       4.63360522e-02, 1.32176183e-01, 4.88580750e-02, 7.11876020e-02,
```

```
       3.06590538e-01, 1.79794454e-01, 5.28874388e-03, 3.13768352e-02,
       3.79543230e-02, 1.38396411e-01, 1.81830343e-02, 2.65470799e-01,
       7.91275693e-02, 5.34218597e-02, 4.90062936e+00, 5.26750408e+01,
       2.82203915e+02]), variance=array([1.07094140e-01, 1.88742036e+00,
2.34317437e-01, 1.78161723e+00,
       4.40325719e-01, 6.79193461e-02, 1.39844435e-01, 1.72001423e-01,
       6.97247542e-02, 4.69800274e-01, 3.58302179e-02, 7.59167719e-01,
       9.28365241e-02, 8.26118648e-02, 7.00470321e-02, 4.29393369e-01,
       2.00636301e-01, 2.92991898e-01, 3.18992370e+00, 1.65626303e-01,
       1.44315254e+00, 1.01505046e+00, 1.19749530e-01, 1.43862796e-01,
       2.45800502e+00, 7.38036013e-01, 1.13920029e+01, 2.31010973e-01,
       4.31507668e-01, 1.90528093e-01, 1.24671084e-01, 1.07425177e-01,
       2.95159161e-01, 1.07745599e-01, 3.08154062e-01, 1.67896547e-01,
       1.85791650e-01, 4.34829439e-02, 1.42525114e-01, 1.16865102e-01,
       1.50361473e-01, 6.09903912e-01, 5.73945833e-02, 3.19259425e-01,
       1.01935877e+00, 8.17471270e-01, 4.63438951e-03, 7.50333517e-02,
       5.54612799e-02, 7.77968333e-02, 1.48045497e-02, 7.59181612e-01,
       6.74541224e-02, 2.69600271e-01, 7.42311765e+02, 4.86573219e+04,
       3.68952901e+05]), skewness=array([ 5.92257918,  9.5555492 ,  2.94110789,
27.15035267,  4.22000271,
        4.55490419,  6.21454549, 10.63604439,  4.44795353,  9.63368819,
        5.1601559 ,  3.12797362,  7.99555783, 10.07103212,  6.44051978,
        5.9017492 ,  5.71193665,  5.63845456,  1.6918398 ,  8.05102821,
        2.36131511,  9.70708774,  5.74851972, 13.62929854,  5.51200726,
        5.77490458,  5.72163481,  5.84582426, 11.30526457,  6.67894971,
        8.78006633, 10.35563132, 16.1291286 , 10.31146394, 17.98980105,
        7.86085564,  5.29526945, 27.69555992, 10.51869112,  9.12514394,
       12.60532735,  9.42688905,  7.88762618, 19.69945392,  9.63372543,
        8.97501221, 18.94255005, 20.98217881, 14.12336521, 16.36382061,
       21.32440567, 21.32959254, 10.88427173, 26.25786993, 27.34951229,
       31.14016596,  9.80477376]), kurtosis=array([  51.71558405,   93.89016173,
13.18839908,  785.40163828,
         28.69487647,   31.20576951,   66.53150801,  198.68010939,
         28.29530115,  185.40607771,   34.48800593,   15.18712484,
        109.66544541,  138.05561341,   44.19188958,   55.62892   ,
         47.49151277,   52.75647121,    6.32523058,   77.87379384,
          8.48736408,  103.7022867 ,   49.37553046,  272.09125904,
         42.43992409,   49.41302953,   33.63974328,   39.86629858,
        166.19735746,   53.12216402,   91.72439904,  124.79234055,
        433.42661801,  123.97955409,  555.16708959,   86.72460731,
         43.92486688,  865.39968623,  181.33012173,  100.87592785,
        189.11563172,  111.21705016,   81.96093958,  567.75150773,
        147.5283386 ,  107.79164424,  445.8361165 ,  634.57001982,
        228.75884956,  499.07842266,  588.19774644,  688.05527222,
        184.31757803,  851.48819158,  954.59095344, 1348.49464105,
        183.78053905]))
```

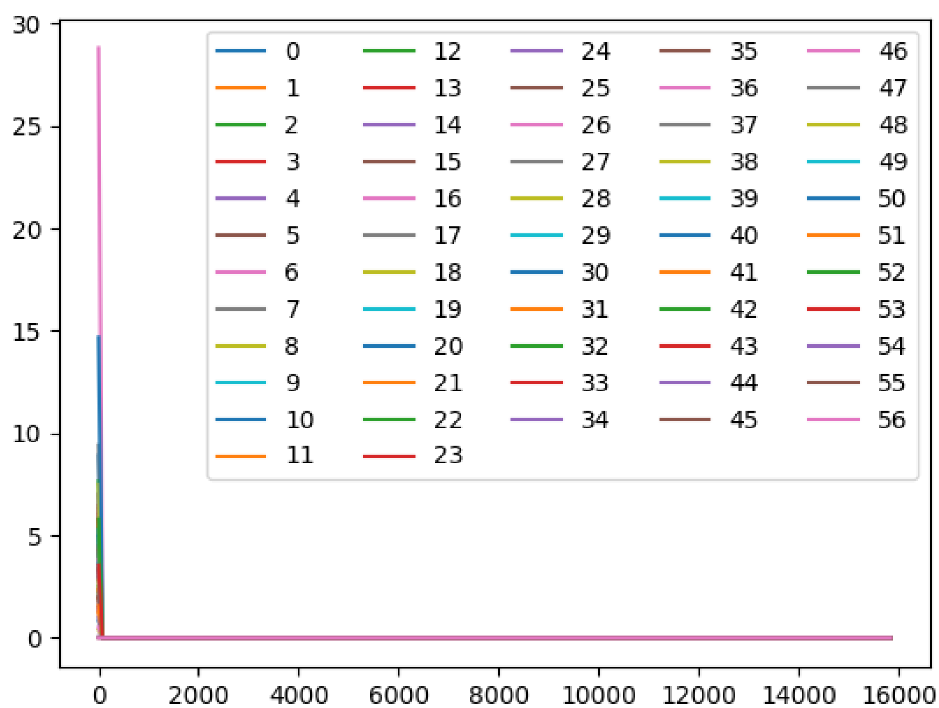
[108]:
```
scipy.stats.describe(y)
```

[108]:
```
DescribeResult(nobs=3065, minmax=(0, 1), mean=0.39738988580750406,
variance=0.23954932085067235, skewness=0.41936632478193103,
kurtosis=-1.824131885638896)
```

2.1 1a) Distributions of all features (unnormalized)

[109]:
```
# plot the distribution of all features
nextplot()
densities = [scipy.stats.gaussian_kde(X[:, j]) for j in range(D)]
xs = np.linspace(np.min(X), np.max(X), 200)
for j in range(D):
    plt.plot(xs, densities[j](xs), label=j)
plt.legend(ncol=5)
```

[109]:
```
<matplotlib.legend.Legend at 0x364b9a650>
```

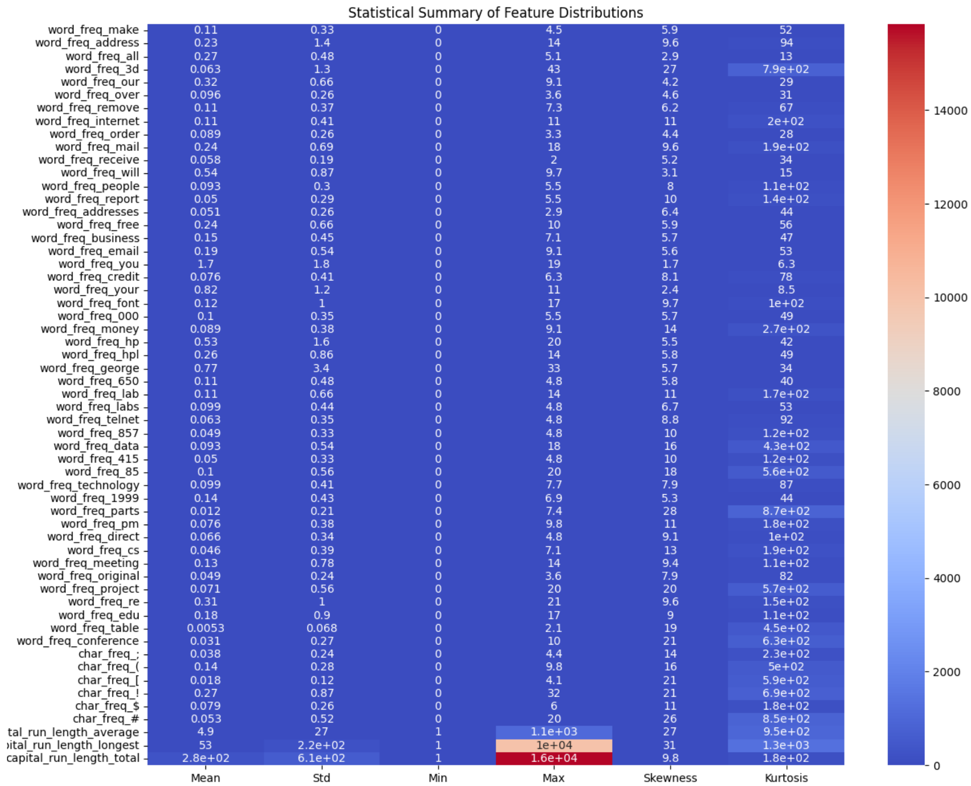


2.1.1 Statistical Summary of Feature Distributions (unnormalized)

[110]:
```python
# this plots is not really helpful; go now explore further
import numpy as np
import scipy.stats
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Collect the statistical summary for each feature
data = []
for j in range(D):
    feature_values = X[:, j]
    mean = np.mean(feature_values)
    std_dev = np.std(feature_values)
    min_val = np.min(feature_values)
    max_val = np.max(feature_values)
    skewness = scipy.stats.skew(feature_values)
    kurtosis = scipy.stats.kurtosis(feature_values)

    data.append([j, mean, std_dev, min_val, max_val, skewness, kurtosis])

# Create a DataFrame
df = pd.DataFrame(data, columns=['Feature', 'Mean', 'Std', 'Min', 'Max',␣
↪'Skewness', 'Kurtosis'])

# Display the DataFrame
print(df)

# Plot the heatmap
plt.figure(figsize=(14, 12))
sns.heatmap(df.iloc[:, 1:], annot=True, cmap='coolwarm', xticklabels=df.
↪columns[1:], yticklabels=features)
plt.title('Statistical Summary of Feature Distributions')
plt.show()
```

```
    Feature      Mean       Std  Min     Max   Skewness    Kurtosis
0         0  0.110819  0.327199  0.0   4.540   5.922579   51.715584
1         1  0.228486  1.373610  0.0  14.280   9.555549   93.890162
2         2  0.274153  0.483984  0.0   5.100   2.941108   13.188399
3         3  0.062969  1.334555  0.0  42.810  27.150353  785.401638
4         4  0.317788  0.663462  0.0   9.090   4.220003   28.694876
5         5  0.095755  0.260571  0.0   3.570   4.554904   31.205770
6         6  0.113546  0.373897  0.0   7.270   6.214545   66.531508
7         7  0.107217  0.414663  0.0  11.110  10.636044  198.680109
8         8  0.088923  0.264011  0.0   3.330   4.447954   28.295301
9         9  0.241719  0.685308  0.0  18.180   9.633688  185.406078
10       10  0.058131  0.189258  0.0   2.000   5.160156   34.488006
```

11	11	0.537432	0.871160	0.0	9.670	3.127974	15.187125
12	12	0.092623	0.304641	0.0	5.550	7.995558	109.665445
13	13	0.049664	0.287376	0.0	5.550	10.071032	138.055613
14	14	0.050721	0.264621	0.0	2.860	6.440520	44.191890
15	15	0.235334	0.655174	0.0	10.160	5.901749	55.628920
16	16	0.147197	0.447851	0.0	7.140	5.711937	47.491513
17	17	0.186600	0.541199	0.0	9.090	5.638455	52.756471
18	18	1.661210	1.785744	0.0	18.750	1.691840	6.325231
19	19	0.076307	0.406906	0.0	6.320	8.051028	77.873794
20	20	0.819592	1.201117	0.0	11.110	2.361315	8.487364
21	21	0.122728	1.007333	0.0	17.100	9.707088	103.702287
22	22	0.102007	0.345992	0.0	5.450	5.748520	49.375530
23	23	0.089080	0.379231	0.0	9.090	13.629299	272.091259
24	24	0.529801	1.567547	0.0	20.000	5.512007	42.439924
25	25	0.262072	0.858950	0.0	14.280	5.774905	49.413030
26	26	0.771507	3.374653	0.0	33.330	5.721635	33.639743
27	27	0.114323	0.480558	0.0	4.760	5.845824	39.866299
28	28	0.109488	0.656785	0.0	14.280	11.305265	166.197357
29	29	0.099295	0.436424	0.0	4.760	6.678950	53.122164
30	30	0.062816	0.353030	0.0	4.760	8.780066	91.724399
31	31	0.049034	0.327704	0.0	4.760	10.355631	124.792341
32	32	0.092747	0.543197	0.0	18.180	16.129129	433.426618
33	33	0.049602	0.328193	0.0	4.760	10.311464	123.979554
34	34	0.102157	0.555026	0.0	20.000	17.989801	555.167090
35	35	0.099305	0.409685	0.0	7.690	7.860856	86.724607
36	36	0.143285	0.430965	0.0	6.890	5.295269	43.924867
37	37	0.012427	0.208492	0.0	7.400	27.695560	865.399686
38	38	0.075592	0.377463	0.0	9.750	10.518691	181.330122
39	39	0.066046	0.341800	0.0	4.760	9.125144	100.875928
40	40	0.046336	0.387701	0.0	7.140	12.605327	189.115632
41	41	0.132176	0.780836	0.0	14.280	9.426889	111.217050
42	42	0.048858	0.239533	0.0	3.570	7.887626	81.960940
43	43	0.071188	0.564938	0.0	20.000	19.699454	567.751508
44	44	0.306591	1.009468	0.0	21.420	9.633725	147.528339
45	45	0.179794	0.903994	0.0	16.700	8.975012	107.791644
46	46	0.005289	0.068065	0.0	2.120	18.942550	445.836117
47	47	0.031377	0.273877	0.0	10.000	20.982179	634.570020
48	48	0.037954	0.235464	0.0	4.385	14.123365	228.758850
49	49	0.138396	0.278875	0.0	9.752	16.363821	499.078423
50	50	0.018183	0.121654	0.0	4.081	21.324406	588.197746
51	51	0.265471	0.871168	0.0	32.478	21.329593	688.055272
52	52	0.079128	0.259677	0.0	6.003	10.884272	184.317578
53	53	0.053422	0.519146	0.0	19.829	26.257870	851.488192
54	54	4.900629	27.240954	1.0	1102.500	27.349512	954.590953
55	55	52.675041	220.548060	1.0	9989.000	31.140166	1348.494641
56	56	282.203915	607.315836	1.0	15841.000	9.804774	183.780539

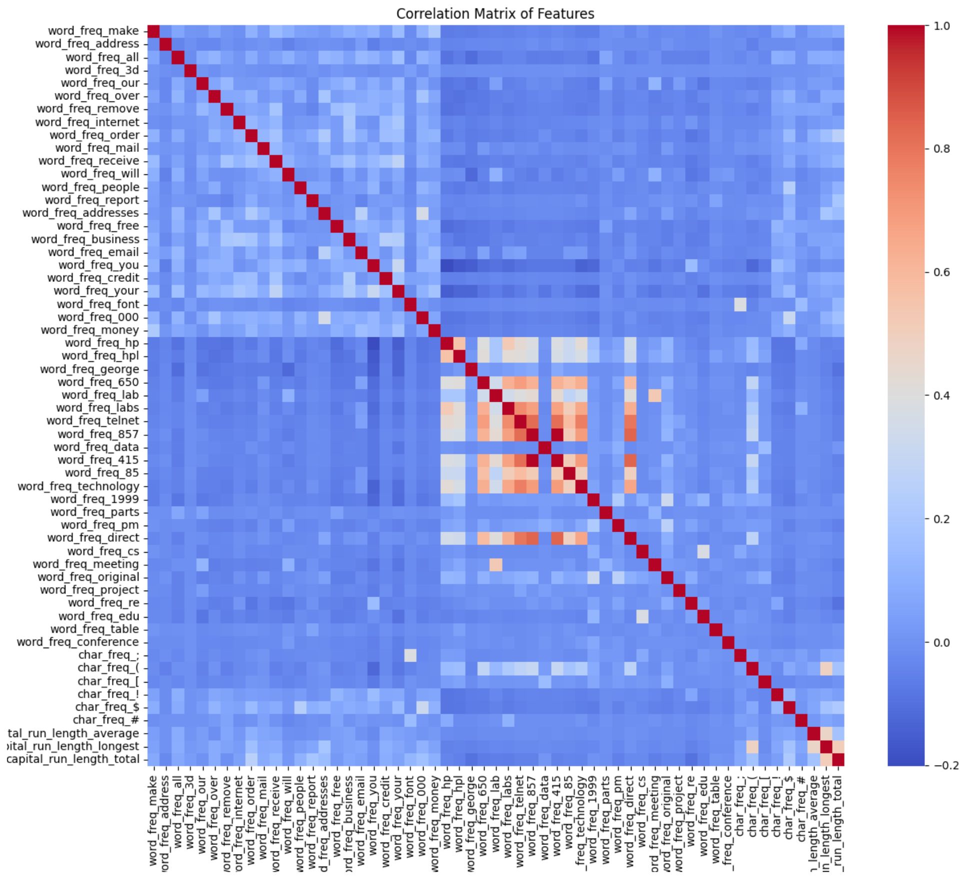


2.1.2 Correlation Matrix of Features

[111]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Create a DataFrame for the feature values
df_features = pd.DataFrame(X, columns=features)

# Compute the correlation matrix
corr_matrix = df_features.corr()

# Find highly correlated feature pairs
threshold = 0.7
highly_correlated_pairs = []
for i in range(len(corr_matrix.columns)):
```

```python
    for j in range(i):
        if abs(corr_matrix.iloc[i, j]) > threshold:
            pair = (corr_matrix.columns[i], corr_matrix.columns[j], corr_matrix.
 ↪iloc[i, j])
            highly_correlated_pairs.append(pair)

# Output the highly correlated feature pairs
print("Highly Correlated Feature Pairs (Threshold > 0.7):")
for pair in highly_correlated_pairs:
    print(f"{pair[0]} and {pair[1]}: {pair[2]:.2f}")

# Plot the heatmap for the correlation matrix without annotations
plt.figure(figsize=(14, 12))
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm', xticklabels=features,␣
 ↪yticklabels=features)
plt.title('Correlation Matrix of Features')
plt.show()
```

```
Highly Correlated Feature Pairs (Threshold > 0.7):
word_freq_telnet and word_freq_labs: 0.74
word_freq_857 and word_freq_telnet: 0.82
word_freq_415 and word_freq_telnet: 0.82
word_freq_415 and word_freq_857: 1.00
word_freq_technology and word_freq_telnet: 0.77
word_freq_direct and word_freq_telnet: 0.79
word_freq_direct and word_freq_857: 0.84
word_freq_direct and word_freq_415: 0.84
```



2.2 1b) Feature Normalization

[112]:
```
# Let's compute z-scores; create two new variables Xz and Xtestz.
# for each feature, we will need mean and standard deviation
means_train = np.mean(X, axis=0)
stds_train = np.std(X, axis=0)

# apply Z-score normalization accroding to the formula
Xz = (X - means_train) / stds_train

# we normalize test data with statistics calculated on train set
Xtestz = (Xtest - means_train) / stds_train
```

[113]:
```
# Let's check. Xz and Xtestz refer to the normalized datasets just created. We
# will use them throughout.
print("Mean of Xz (should be all 0):")
print(np.mean(Xz, axis=0))
```

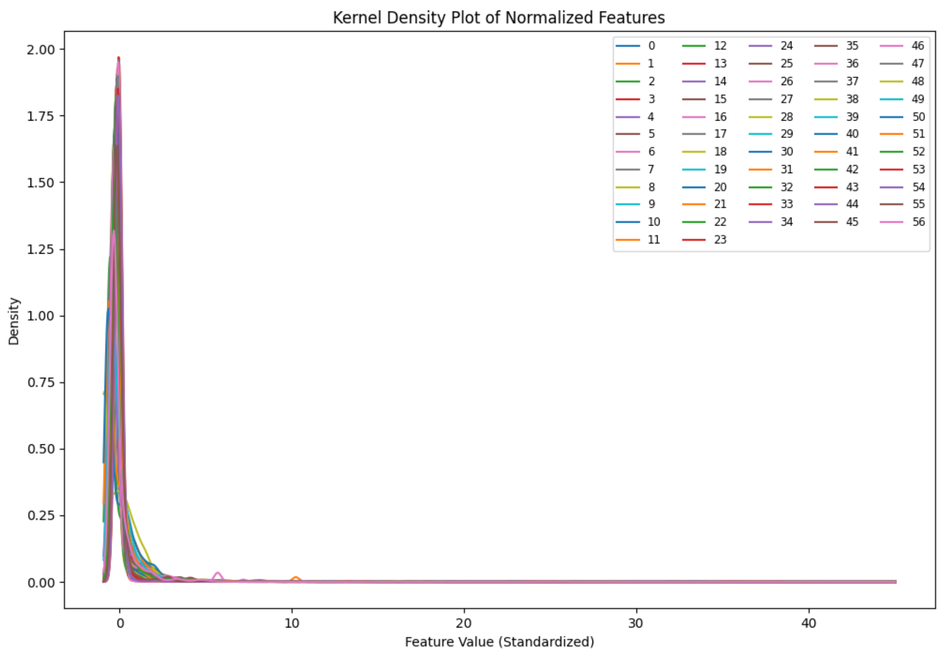
```python
print("\nVariance of Xz (should be all 1):")
print(np.var(Xz, axis=0))

print("\nMean of Xtestz:")
print(np.mean(Xtestz, axis=0))

print("\nVariance of Xtestz:")
print(np.var(Xtestz, axis=0))

print("\nSum of Xz^3 (should be: 1925261.15):")
print(np.sum(Xz ** 3))
```

```
Mean of Xz (should be all 0):
[ 1.85459768e-17  9.27298839e-18 -5.56379304e-17 -9.27298839e-18
  5.56379304e-17  3.70919536e-17  0.00000000e+00 -7.41839072e-17
  5.56379304e-17  0.00000000e+00 -1.85459768e-17 -2.43415945e-17
 -4.63649420e-17  1.85459768e-17  1.85459768e-17  3.70919536e-17
 -3.70919536e-17 -9.27298839e-17 -1.66913791e-16  9.27298839e-18
  1.85459768e-17  9.27298839e-18 -5.56379304e-17 -1.85459768e-17
 -6.49109188e-17 -3.70919536e-17 -1.85459768e-17  1.85459768e-17
 -2.78189652e-17  4.63649420e-17 -1.85459768e-17  5.56379304e-17
  0.00000000e+00 -1.85459768e-17  3.70919536e-17  1.85459768e-17
 -9.27298839e-18  4.63649420e-18  1.85459768e-17  9.27298839e-18
  2.31824710e-17 -2.78189652e-17 -9.27298839e-18  4.63649420e-18
 -9.27298839e-18 -9.27298839e-18  1.39094826e-17 -2.78189652e-17
 -3.70919536e-17 -6.49109188e-17  4.63649420e-18  3.70919536e-17
 -3.70919536e-17  9.27298839e-18 -9.27298839e-18  9.27298839e-18
 -7.41839072e-17]

Variance of Xz (should be all 1):
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Mean of Xtestz:
[-5.73600192e-02 -3.37389835e-02  4.02481250e-02  5.51233798e-03
 -2.51229644e-02  1.67364997e-03  5.29785531e-03 -1.38875040e-02
  1.29802458e-02 -1.00804532e-02  2.68026912e-02  1.46804853e-02
  1.28455840e-02  9.34193448e-02 -1.71666713e-02  6.17841473e-02
 -3.08405298e-02 -1.02710095e-02  1.49139906e-03  6.82438979e-02
 -2.45179646e-02 -4.53675036e-03 -3.12737328e-03  4.09841941e-02
  3.76515934e-02  1.15494599e-02 -3.73018154e-03  6.55839018e-02
 -4.82178216e-02  2.44089391e-02  1.64408852e-02 -1.81514851e-02
  2.47142980e-02 -1.61248615e-02  1.75684573e-02 -1.33686432e-02
 -4.40153254e-02  1.11212504e-02  2.40959269e-02 -1.06211719e-02
 -2.06246544e-02  6.23149655e-04 -3.45073187e-02  4.24615929e-02
```

```
 -1.59254291e-02  9.77429328e-05  6.85319587e-03  5.38462415e-03
  7.89156240e-03  6.81007462e-03 -2.97234292e-02  1.23785037e-02
 -3.82610483e-02 -5.29891640e-02  3.19860888e-02 -6.82149671e-03
  5.35333143e-03]

Variance of Xtestz:
[0.61068019 0.64746339 1.25293677 1.2774661  1.08119249 1.31173762
 1.28697678 0.80611698 1.33973062 0.65533893 1.40034314 0.93450565
 0.92877323 2.0728468  0.86981179 2.75968123 0.94816223 0.88879741
 0.96502082 2.70171906 0.99741759 1.1098788  1.07414603 2.08336518
 1.40816544 1.19772845 0.9862879  1.76326753 0.44704368 1.28342341
 1.91457064 1.01476883 1.14073258 1.02208023 0.75850361 0.89687605
 0.89454052 1.35876298 1.97554069 1.14319113 0.60370645 0.89279613
 0.61835224 1.633395   1.01236044 1.04674566 1.76525404 1.2642542
 1.20646248 0.81912474 0.42556335 0.62984245 0.68863812 0.05099329
 2.06687781 0.34306778 0.98979083]

Sum of Xz^3 (should be: 1925261.15):
1925261.1560010156
```

2.3 1c) Distributions of all features (normalized)

[114]:
```python
# Explore the normalized data
import matplotlib.pyplot as plt
import scipy.stats

# Set up the plot
plt.figure(figsize=(12, 8))
densities_z = [scipy.stats.gaussian_kde(Xz[:, j]) for j in range(Xz.shape[1])]
xs = np.linspace(np.min(Xz), np.max(Xz), 1000)

# Plot each feature's density
for j in range(Xz.shape[1]):
    plt.plot(xs, densities_z[j](xs), label=j)

plt.legend(ncol=5, fontsize='small')
plt.title('Kernel Density Plot of Normalized Features')
plt.xlabel('Feature Value (Standardized)')
plt.ylabel('Density')
plt.show()
```



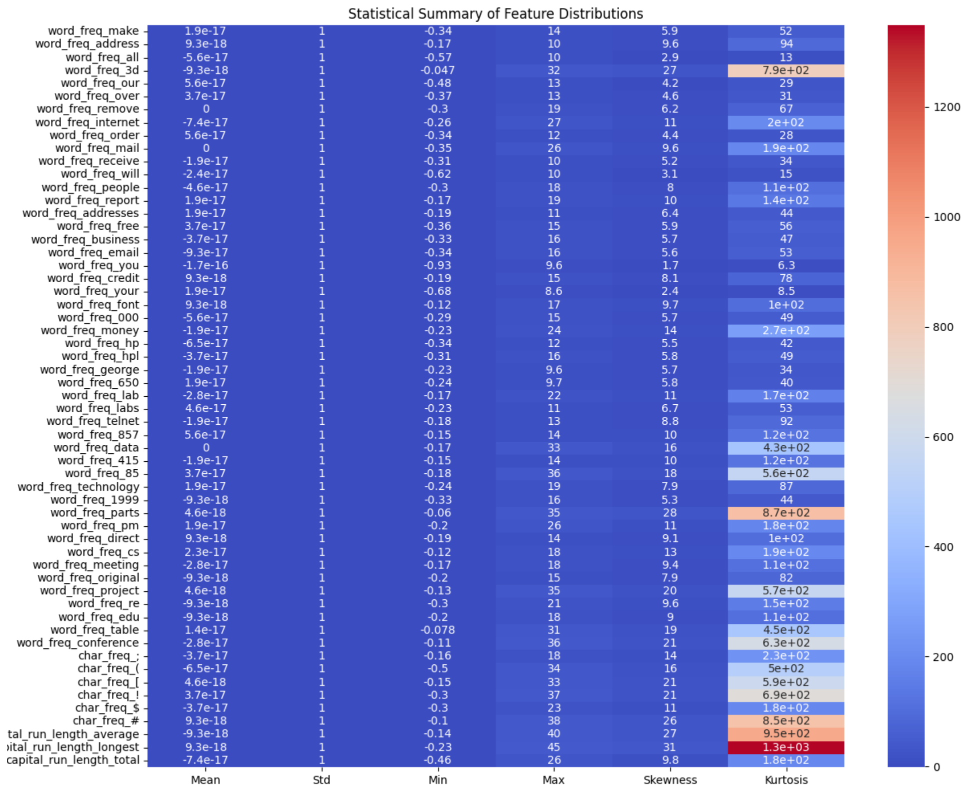
2.3.1 Statistical Summary of Feature Distributions (normalized)

[115]:
```
# this plots is not really helpful; go now explore further
import numpy as np
import scipy.stats
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Collect the statistical summary for each feature
data = []
for j in range(D):
    feature_values = Xz[:, j]
    mean = np.mean(feature_values)
    std_dev = np.std(feature_values)
    min_val = np.min(feature_values)
    max_val = np.max(feature_values)
    skewness = scipy.stats.skew(feature_values)
    kurtosis = scipy.stats.kurtosis(feature_values)

    data.append([j, mean, std_dev, min_val, max_val, skewness, kurtosis])

# Create a DataFrame
```

```python
df = pd.DataFrame(data, columns=['Feature', 'Mean', 'Std', 'Min', 'Max', 'Skewness', 'Kurtosis'])

# Display the DataFrame
print(df)

# Plot the heatmap
plt.figure(figsize=(14, 12))
sns.heatmap(df.iloc[:, 1:], annot=True, cmap='coolwarm', xticklabels=df.columns[1:], yticklabels=features)
plt.title('Statistical Summary of Feature Distributions')
plt.show()
```

```
    Feature          Mean  Std       Min        Max   Skewness    Kurtosis
0         0  1.854598e-17  1.0 -0.338690  13.536658   5.922579   51.715584
1         1  9.272988e-18  1.0 -0.166340  10.229624   9.555549   93.890162
2         2 -5.563793e-17  1.0 -0.566451   9.971077   2.941108   13.188399
3         3 -9.272988e-18  1.0 -0.047184  32.030935  27.150353  785.401638
4         4  5.563793e-17  1.0 -0.478984  13.221872   4.220003   28.694876
5         5  3.709195e-17  1.0 -0.367483  13.333204   4.554904   31.205770
6         6  0.000000e+00  1.0 -0.303684  19.140184   6.214545   66.531508
7         7 -7.418391e-17  1.0 -0.258564  26.534285  10.636044  198.680109
8         8  5.563793e-17  1.0 -0.336816  12.276277   4.447954   28.295301
9         9  0.000000e+00  1.0 -0.352716  26.175503   9.633688  185.406078
10       10 -1.854598e-17  1.0 -0.307150  10.260444   5.160156   34.488006
11       11 -2.434159e-17  1.0 -0.616916  10.483225   3.127974   15.187125
12       12 -4.636494e-17  1.0 -0.304040  17.914115   7.995558  109.665445
13       13  1.854598e-17  1.0 -0.172819  19.139865  10.071032  138.055613
14       14  1.854598e-17  1.0 -0.191674  10.616243   6.440520   44.191890
15       15  3.709195e-17  1.0 -0.359194  15.148132   5.901749   55.628920
16       16 -3.709195e-17  1.0 -0.328675  15.614115   5.711937   47.491513
17       17 -9.272988e-17  1.0 -0.344791  16.451251   5.638455   52.756471
18       18 -1.669138e-16  1.0 -0.930262   9.569561   1.691840    6.325231
19       19  9.272988e-18  1.0 -0.187529  15.344324   8.051028   77.873794
20       20  1.854598e-17  1.0 -0.682358   8.567366   2.361315    8.487364
21       21  9.272988e-18  1.0 -0.121834  16.853688   9.707088  103.702287
22       22 -5.563793e-17  1.0 -0.294823  15.456986   5.748520   49.375530
23       23 -1.854598e-17  1.0 -0.234896  23.734687  13.629299  272.091259
24       24 -6.491092e-17  1.0 -0.337981  12.420809   5.512007   42.439924
25       25 -3.709195e-17  1.0 -0.305107  16.319841   5.774905   49.413030
26       26 -1.854598e-17  1.0 -0.228618   9.647951   5.721635   33.639743
27       27  1.854598e-17  1.0 -0.237897   9.667264   5.845824   39.866299
28       28 -2.781897e-17  1.0 -0.166703  21.575564  11.305265  166.197357
29       29  4.636494e-17  1.0 -0.227520  10.679304   6.678950   53.122164
30       30 -1.854598e-17  1.0 -0.177933  13.305328   8.780066   91.724399
31       31  5.563793e-17  1.0 -0.149630  14.375659  10.355631  124.792341
32       32  0.000000e+00  1.0 -0.170743  33.297784  16.129129  433.426618
```

33	33	-1.854598e-17	1.0	-0.151137	14.352538	10.311464	123.979554
34	34	3.709195e-17	1.0	-0.184057	35.850310	17.989801	555.167090
35	35	1.854598e-17	1.0	-0.242394	18.528127	7.860856	86.724607
36	36	-9.272988e-18	1.0	-0.332476	15.654893	5.295269	43.924867
37	37	4.636494e-18	1.0	-0.059606	35.433426	27.695560	865.399686
38	38	1.854598e-17	1.0	-0.200264	25.630056	10.518691	181.330122
39	39	9.272988e-18	1.0	-0.193229	13.733059	9.125144	100.875928
40	40	2.318247e-17	1.0	-0.119515	18.296718	12.605327	189.115632
41	41	-2.781897e-17	1.0	-0.169275	18.118815	9.426889	111.217050
42	42	-9.272988e-18	1.0	-0.203973	14.700054	7.887626	81.960940
43	43	4.636494e-18	1.0	-0.126010	35.276088	19.699454	567.751508
44	44	-9.272988e-18	1.0	-0.303715	20.915377	9.633725	147.528339
45	45	-9.272988e-18	1.0	-0.198889	18.274692	8.975012	107.791644
46	46	1.390948e-17	1.0	-0.077701	31.068885	18.942550	445.836117
47	47	-2.781897e-17	1.0	-0.114565	36.398113	20.982179	634.570020
48	48	-3.709195e-17	1.0	-0.161190	18.461633	14.123365	228.758850
49	49	-6.491092e-17	1.0	-0.496266	34.472764	16.363821	499.078423
50	50	4.636494e-18	1.0	-0.149465	33.396466	21.324406	588.197746
51	51	3.709195e-17	1.0	-0.304730	36.976248	21.329593	688.055272
52	52	-3.709195e-17	1.0	-0.304715	22.812470	10.884272	184.317578
53	53	9.272988e-18	1.0	-0.102903	38.092536	26.257870	851.488192
54	54	-9.272988e-18	1.0	-0.143190	40.292252	27.349512	954.590953
55	55	9.272988e-18	1.0	-0.234303	45.052879	31.140166	1348.494641
56	56	-7.418391e-17	1.0	-0.463027	25.618953	9.804774	183.780539



3 2. Maximum Likelihood Estimation

3.1 Helper functions

[116]:
```python
def logsumexp(x):
    """Computes log(sum(exp(x)).

    Uses offset trick to reduce risk of numeric over- or underflow. When x is a
    1D ndarray, computes logsumexp of its entries. When x is a 2D ndarray,
    computes logsumexp of each column.

    Keyword arguments:
    x : a 1D or 2D ndarray
    """
    offset = np.max(x, axis=0)
    return offset + np.log(np.sum(np.exp(x - offset), axis=0))
```

[117]:
```python
# Define the logistic function. Make sure it operates on both scalars
# and vectors.
def sigma(x):
    return 1 / (1 + np.exp(-x))
```

[118]:
```python
# this should give:
# [0.5, array([0.26894142, 0.5, 0.73105858])]
[sigma(0), sigma(np.array([-1, 0, 1]))]
```

[118]:
```
[0.5, array([0.26894142, 0.5       , 0.73105858])]
```

[119]:
```python
# Define the logarithm of the logistic function. Make sure it operates on both
# scalars and vectors. Perhaps helpful: isinstance(x, np.ndarray).
def logsigma(x):
    return np.log(sigma(x))
```

[120]:
```python
# this should give:
# [-0.69314718055994529, array([-1.31326169, -0.69314718, -0.31326169])]
[logsigma(0), logsigma(np.array([-1, 0, 1]))]
```

[120]:
```
[-0.6931471805599453, array([-1.31326169, -0.69314718, -0.31326169])]
```

3.2 2b Log-likelihood and gradient

[121]:
```python
def l(y, X, w):
    """Log-likelihood of the logistic regression model.

    Parameters
    ----------
    y : ndarray of shape (N,)
        Binary labels (either 0 or 1).
    X : ndarray of shape (N,D)
        Design matrix.
    w : ndarray of shape (D,)
        Weight vector.
    """
    log_likelihood = np.sum(y * logsigma( X @ w) + (1-y) * logsigma(- X @ w))

    return log_likelihood
```

[122]:
```python
# this should give:
# -47066.641667825766
l(y, Xz, np.linspace(-5, 5, D))
```

[122]:
```
-47066.641667825774
```

[123]:
```python
def dl(y, X, w):
    """Gradient of the log-likelihood of the logistic regression model.

    Parameters
    ----------
    y : ndarray of shape (N,)
        Binary labels (either 0 or 1).
    X : ndarray of shape (N,D)
        Design matrix.
    w : ndarray of shape (D,)
        Weight vector.

    Returns
    -------
    ndarray of shape (D,)
    """

    # getting the error vector
    e = y - sigma(X @ w)


    # computing the gradient
    gradient = e.T @ X
    #gradient = X.T @ e

    return gradient
```

[124]:
```python
# this should give:
# array([  551.33985842,   143.84116318,   841.83373606,   156.87237578,
#          802.61217579,   795.96202907,   920.69045803,   621.96516752,
#          659.18724769,   470.81259805,   771.32406968,   352.40325626,
#          455.66972482,   234.36600888,   562.45454038,   864.83981264,
#          787.19723703,   649.48042176,   902.6478154 ,   544.00539886,
#         1174.78638035,   120.3598967 ,   839.61141672,   633.30453444,
#         -706.66815087,  -630.2039816 ,  -569.3451386 ,  -527.50996698,
#         -359.53701083,  -476.64334832,  -411.60620464,  -375.11950586,
#         -345.37195689,  -376.22044258,  -407.31761977,  -456.23251936,
#         -596.86960184,  -107.97072355,  -394.82170044,  -229.18125598,
#         -288.46356547,  -362.13402385,  -450.87896465,  -277.03932676,
#         -414.99293368,  -452.28771693,  -167.54649092,  -270.9043748 ,
#         -252.20140951,  -357.72497343,  -259.12468742,   418.35938483,
#          604.54173228,    43.10390907,   152.24258478,   378.16731033,
#          416.12032881])
dl(y, Xz, np.linspace(-5, 5, D))
```

[124]:
```
array([ 551.33985842,  143.84116318,  841.83373606,  156.87237578,
        802.61217579,  795.96202907,  920.69045803,  621.96516752,
```

```
 659.18724769,  470.81259805,  771.32406968,  352.40325626,
 455.66972482,  234.36600888,  562.45454038,  864.83981264,
 787.19723703,  649.48042176,  902.6478154 ,  544.00539886,
1174.78638035,  120.3598967 ,  839.61141672,  633.30453444,
-706.66815087, -630.2039816 , -569.3451386 , -527.50996698,
-359.53701083, -476.64334832, -411.60620464, -375.11950586,
-345.37195689, -376.22044258, -407.31761977, -456.23251936,
-596.86960184, -107.97072355, -394.82170044, -229.18125598,
-288.46356547, -362.13402385, -450.87896465, -277.03932676,
-414.99293368, -452.28771693, -167.54649092, -270.9043748 ,
-252.20140951, -357.72497343, -259.12468742,  418.35938483,
 604.54173228,   43.10390907,  152.24258478,  378.16731033,
 416.12032881])
```

3.3 2c Gradient descent

[125]:
```python
# you don't need to modify this function
def optimize(obj_up, theta0, nepochs=50, eps0=0.01, verbose=True):
    """Iteratively minimize a function.

    We use it here to run either gradient descent or stochastic gradient
    descent, using arbitrarly optimization criteria.

    Parameters
    ----------
    obj_up  : a tuple of form (f, update) containing two functions f and update.
              f(theta) computes the value of the objective function.
              update(theta,eps) performs an epoch of parameter update with step␣
 ↪size
              eps and returns the result.
    theta0  : ndarray of shape (D,)
              Initial parameter vector.
    nepochs : int
              How many epochs (calls to update) to run.
    eps0    : float
              Initial step size.
    verbose : boolean
              Whether to print progress information.

    Returns
    -------
    A triple consisting of the fitted parameter vector, the values of the
    objective function after every epoch, and the step sizes that were used.
    """

    f, update = obj_up
```

```
    # initialize results
    theta = theta0
    values = np.zeros(nepochs + 1)
    eps = np.zeros(nepochs + 1)
    values[0] = f(theta0)
    eps[0] = eps0

    # now run the update function nepochs times
    for epoch in range(nepochs):
        if verbose:
            print(
                "Epoch {:3d}: f={:10.3f}, eps={:10.9f}".format(
                    epoch, values[epoch], eps[epoch]
                )
            )
        theta = update(theta, eps[epoch])

        # we use the bold driver heuristic
        values[epoch + 1] = f(theta)
        if values[epoch] < values[epoch + 1]:
            eps[epoch + 1] = eps[epoch] / 2.0
        else:
            eps[epoch + 1] = eps[epoch] * 1.05

    # all done
    if verbose:
        print("Result after {} epochs: f={}".format(nepochs, values[-1]))
    return theta, values, eps
```

[126]:
```
# define the objective and update function for one gradient-descent epoch for
# fitting an MLE estimate of logistic regression with gradient descent (should
# return a tuple of two functions; see optimize)
def gd(y, X):
    def objective(w):
        # computing the negative log likelihood
        return -l(y, X, w)

    def update(w, eps):
        # computing the update
        return w + eps * dl(y, X, w)
    return (objective, update)
```

[127]:
```
# this should give
# [47066.641667825766,
#  array([ 4.13777838e+01,  -1.56745627e+01,   5.75882538e+01,
#          1.14225143e+01,   5.54249703e+01,   5.99229049e+01,
#          7.11220141e+01,   4.84761728e+01,   5.78067289e+01,
```

```
#           4.54794720e+01,   7.14638492e+01,   1.51369386e+01,
#           3.36375739e+01,   2.15061217e+01,   5.78014255e+01,
#           6.72743066e+01,   7.00829312e+01,   5.29328088e+01,
#           6.16042473e+01,   5.50018510e+01,   8.94624817e+01,
#           2.74784480e+01,   8.51763599e+01,   5.60363965e+01,
#          -2.55865589e+01,  -1.53788213e+01,  -4.67015412e+01,
#          -2.50356570e+00,  -3.85357592e+00,  -2.21819155e+00,
#           3.32098671e+00,   3.86933390e+00,  -2.00309898e+01,
#           3.84684492e+00,  -2.19847927e-01,  -1.29775457e+00,
#          -1.28374302e+01,  -2.78303173e+00,  -5.61671182e+00,
#           1.73657121e+01,  -6.81197570e+00,  -1.20249002e+01,
#           2.65789491e+00,  -1.39557852e+01,  -2.01135653e+01,
#          -2.72134051e+01,  -9.45952961e-01,  -1.02239111e+01,
#           1.52794293e-04,  -5.18938123e-01,  -3.19717561e+00,
#           4.62953437e+01,   7.87893022e+01,   1.88618651e+01,
#           2.85195027e+01,   5.04698358e+01,   6.41240689e+01])
f, update = gd(y, Xz)
[f(np.linspace(-5, 5, D)), update(np.linspace(-5, -5, D), 0.1)]
```

[127]:

```
[47066.641667825774,
 array([ 4.13777838e+01, -1.56745627e+01,  5.75882538e+01,  1.14225143e+01,
         5.54249703e+01,  5.99229049e+01,  7.11220141e+01,  4.84761728e+01,
         5.78067289e+01,  4.54794720e+01,  7.14638492e+01,  1.51369386e+01,
         3.36375739e+01,  2.15061217e+01,  5.78014255e+01,  6.72743066e+01,
         7.00829312e+01,  5.29328088e+01,  6.16042473e+01,  5.50018510e+01,
         8.94624817e+01,  2.74784480e+01,  8.51763599e+01,  5.60363965e+01,
        -2.55865589e+01, -1.53788213e+01, -4.67015412e+01, -2.50356570e+00,
        -3.85357592e+00, -2.21819155e+00,  3.32098671e+00,  3.86933390e+00,
        -2.00309898e+01,  3.84684492e+00, -2.19847927e-01, -1.29775457e+00,
        -1.28374302e+01, -2.78303173e+00, -5.61671182e+00,  1.73657121e+01,
        -6.81197570e+00, -1.20249002e+01,  2.65789491e+00, -1.39557852e+01,
        -2.01135653e+01, -2.72134051e+01, -9.45952961e-01, -1.02239111e+01,
         1.52794293e-04, -5.18938123e-01, -3.19717561e+00,  4.62953437e+01,
         7.87893022e+01,  1.88618651e+01,  2.85195027e+01,  5.04698358e+01,
         6.41240689e+01])]
```

[128]:

```
# you can run gradient descent!
numpy.random.seed(0)
w0 = np.random.normal(size=D)
wz_gd, vz_gd, ez_gd = optimize(gd(y, Xz), w0, nepochs=500)
```

```
Epoch   0: f=  6636.208, eps=0.010000000
Epoch   1: f=  4216.957, eps=0.010500000
Epoch   2: f=  2657.519, eps=0.011025000
Epoch   3: f=  1926.135, eps=0.011576250
Epoch   4: f=  1449.495, eps=0.012155063
Epoch   5: f=  1207.529, eps=0.012762816
```

```
Epoch   6: f=  1052.489, eps=0.013400956
Epoch   7: f=   957.275, eps=0.014071004
Epoch   8: f=   899.610, eps=0.014774554
Epoch   9: f=   882.904, eps=0.015513282
Epoch  10: f=  1017.083, eps=0.007756641
Epoch  11: f=   840.760, eps=0.008144473
Epoch  12: f=   805.649, eps=0.008551697
Epoch  13: f=   822.108, eps=0.004275848
Epoch  14: f=   746.377, eps=0.004489641
Epoch  15: f=   735.803, eps=0.004714123
Epoch  16: f=   729.780, eps=0.004949829
Epoch  17: f=   724.467, eps=0.005197320
Epoch  18: f=   719.408, eps=0.005457186
Epoch  19: f=   714.564, eps=0.005730046
Epoch  20: f=   709.932, eps=0.006016548
Epoch  21: f=   705.514, eps=0.006317375
Epoch  22: f=   701.321, eps=0.006633244
Epoch  23: f=   697.373, eps=0.006964906
Epoch  24: f=   693.728, eps=0.007313152
Epoch  25: f=   690.591, eps=0.007678809
Epoch  26: f=   688.614, eps=0.008062750
Epoch  27: f=   688.607, eps=0.008465887
Epoch  28: f=   690.854, eps=0.004232944
Epoch  29: f=   679.967, eps=0.004444591
Epoch  30: f=   678.649, eps=0.004666820
Epoch  31: f=   677.447, eps=0.004900161
Epoch  32: f=   676.292, eps=0.005145169
Epoch  33: f=   675.182, eps=0.005402428
Epoch  34: f=   674.120, eps=0.005672549
Epoch  35: f=   673.114, eps=0.005956177
Epoch  36: f=   672.177, eps=0.006253986
Epoch  37: f=   671.334, eps=0.006566685
Epoch  38: f=   670.656, eps=0.006895019
Epoch  39: f=   670.397, eps=0.007239770
Epoch  40: f=   671.342, eps=0.003619885
Epoch  41: f=   668.932, eps=0.003800879
Epoch  42: f=   668.378, eps=0.003990923
Epoch  43: f=   668.027, eps=0.004190469
Epoch  44: f=   667.720, eps=0.004399993
Epoch  45: f=   667.433, eps=0.004619993
Epoch  46: f=   667.159, eps=0.004850992
Epoch  47: f=   666.897, eps=0.005093542
Epoch  48: f=   666.650, eps=0.005348219
Epoch  49: f=   666.417, eps=0.005615630
Epoch  50: f=   666.201, eps=0.005896411
Epoch  51: f=   666.008, eps=0.006191232
Epoch  52: f=   665.858, eps=0.006500794
Epoch  53: f=   665.812, eps=0.006825833
```

```
Epoch  54: f=   666.068, eps=0.003412917
Epoch  55: f=   665.424, eps=0.003583562
Epoch  56: f=   665.290, eps=0.003762741
Epoch  57: f=   665.204, eps=0.003950878
Epoch  58: f=   665.128, eps=0.004148421
Epoch  59: f=   665.054, eps=0.004355843
Epoch  60: f=   664.982, eps=0.004573635
Epoch  61: f=   664.911, eps=0.004802316
Epoch  62: f=   664.842, eps=0.005042432
Epoch  63: f=   664.773, eps=0.005294554
Epoch  64: f=   664.707, eps=0.005559282
Epoch  65: f=   664.641, eps=0.005837246
Epoch  66: f=   664.578, eps=0.006129108
Epoch  67: f=   664.518, eps=0.006435563
Epoch  68: f=   664.467, eps=0.006757341
Epoch  69: f=   664.446, eps=0.007095208
Epoch  70: f=   664.544, eps=0.003547604
Epoch  71: f=   664.339, eps=0.003724984
Epoch  72: f=   664.278, eps=0.003911234
Epoch  73: f=   664.239, eps=0.004106795
Epoch  74: f=   664.206, eps=0.004312135
Epoch  75: f=   664.173, eps=0.004527742
Epoch  76: f=   664.139, eps=0.004754129
Epoch  77: f=   664.106, eps=0.004991835
Epoch  78: f=   664.072, eps=0.005241427
Epoch  79: f=   664.037, eps=0.005503499
Epoch  80: f=   664.002, eps=0.005778674
Epoch  81: f=   663.967, eps=0.006067607
Epoch  82: f=   663.936, eps=0.006370988
Epoch  83: f=   663.918, eps=0.006689537
Epoch  84: f=   663.948, eps=0.003344768
Epoch  85: f=   663.839, eps=0.003512007
Epoch  86: f=   663.807, eps=0.003687607
Epoch  87: f=   663.783, eps=0.003871988
Epoch  88: f=   663.760, eps=0.004065587
Epoch  89: f=   663.737, eps=0.004268866
Epoch  90: f=   663.713, eps=0.004482310
Epoch  91: f=   663.688, eps=0.004706425
Epoch  92: f=   663.661, eps=0.004941746
Epoch  93: f=   663.634, eps=0.005188834
Epoch  94: f=   663.606, eps=0.005448275
Epoch  95: f=   663.576, eps=0.005720689
Epoch  96: f=   663.546, eps=0.006006724
Epoch  97: f=   663.514, eps=0.006307060
Epoch  98: f=   663.482, eps=0.006622413
Epoch  99: f=   663.451, eps=0.006953533
Epoch 100: f=   663.427, eps=0.007301210
Epoch 101: f=   663.442, eps=0.003650605
```

```
Epoch 102: f=   663.371, eps=0.003833135
Epoch 103: f=   663.340, eps=0.004024792
Epoch 104: f=   663.316, eps=0.004226032
Epoch 105: f=   663.294, eps=0.004437333
Epoch 106: f=   663.271, eps=0.004659200
Epoch 107: f=   663.248, eps=0.004892160
Epoch 108: f=   663.223, eps=0.005136768
Epoch 109: f=   663.198, eps=0.005393606
Epoch 110: f=   663.172, eps=0.005663287
Epoch 111: f=   663.146, eps=0.005946451
Epoch 112: f=   663.121, eps=0.006243773
Epoch 113: f=   663.102, eps=0.006555962
Epoch 114: f=   663.108, eps=0.003277981
Epoch 115: f=   663.042, eps=0.003441880
Epoch 116: f=   663.019, eps=0.003613974
Epoch 117: f=   663.001, eps=0.003794673
Epoch 118: f=   662.982, eps=0.003984406
Epoch 119: f=   662.963, eps=0.004183627
Epoch 120: f=   662.943, eps=0.004392808
Epoch 121: f=   662.922, eps=0.004612449
Epoch 122: f=   662.900, eps=0.004843071
Epoch 123: f=   662.877, eps=0.005085225
Epoch 124: f=   662.853, eps=0.005339486
Epoch 125: f=   662.828, eps=0.005606460
Epoch 126: f=   662.802, eps=0.005886783
Epoch 127: f=   662.774, eps=0.006181122
Epoch 128: f=   662.745, eps=0.006490178
Epoch 129: f=   662.715, eps=0.006814687
Epoch 130: f=   662.685, eps=0.007155422
Epoch 131: f=   662.659, eps=0.007513193
Epoch 132: f=   662.656, eps=0.007888852
Epoch 133: f=   662.786, eps=0.003944426
Epoch 134: f=   662.631, eps=0.004141647
Epoch 135: f=   662.578, eps=0.004348730
Epoch 136: f=   662.545, eps=0.004566166
Epoch 137: f=   662.519, eps=0.004794475
Epoch 138: f=   662.497, eps=0.005034198
Epoch 139: f=   662.477, eps=0.005285908
Epoch 140: f=   662.462, eps=0.005550204
Epoch 141: f=   662.457, eps=0.005827714
Epoch 142: f=   662.476, eps=0.002913857
Epoch 143: f=   662.373, eps=0.003059550
Epoch 144: f=   662.355, eps=0.003212527
Epoch 145: f=   662.340, eps=0.003373154
Epoch 146: f=   662.325, eps=0.003541811
Epoch 147: f=   662.310, eps=0.003718902
Epoch 148: f=   662.293, eps=0.003904847
Epoch 149: f=   662.276, eps=0.004100089
```

```
Epoch 150: f=   662.257, eps=0.004305094
Epoch 151: f=   662.238, eps=0.004520348
Epoch 152: f=   662.218, eps=0.004746366
Epoch 153: f=   662.197, eps=0.004983684
Epoch 154: f=   662.175, eps=0.005232868
Epoch 155: f=   662.152, eps=0.005494512
Epoch 156: f=   662.128, eps=0.005769237
Epoch 157: f=   662.103, eps=0.006057699
Epoch 158: f=   662.076, eps=0.006360584
Epoch 159: f=   662.048, eps=0.006678613
Epoch 160: f=   662.019, eps=0.007012544
Epoch 161: f=   661.989, eps=0.007363171
Epoch 162: f=   661.957, eps=0.007731330
Epoch 163: f=   661.924, eps=0.008117896
Epoch 164: f=   661.890, eps=0.008523791
Epoch 165: f=   661.859, eps=0.008949981
Epoch 166: f=   661.868, eps=0.004474990
Epoch 167: f=   661.834, eps=0.004698740
Epoch 168: f=   661.809, eps=0.004933677
Epoch 169: f=   661.791, eps=0.005180361
Epoch 170: f=   661.780, eps=0.005439379
Epoch 171: f=   661.784, eps=0.002719689
Epoch 172: f=   661.698, eps=0.002855674
Epoch 173: f=   661.685, eps=0.002998458
Epoch 174: f=   661.672, eps=0.003148380
Epoch 175: f=   661.659, eps=0.003305799
Epoch 176: f=   661.645, eps=0.003471089
Epoch 177: f=   661.630, eps=0.003644644
Epoch 178: f=   661.615, eps=0.003826876
Epoch 179: f=   661.599, eps=0.004018220
Epoch 180: f=   661.582, eps=0.004219131
Epoch 181: f=   661.564, eps=0.004430087
Epoch 182: f=   661.546, eps=0.004651592
Epoch 183: f=   661.526, eps=0.004884171
Epoch 184: f=   661.506, eps=0.005128380
Epoch 185: f=   661.485, eps=0.005384799
Epoch 186: f=   661.462, eps=0.005654039
Epoch 187: f=   661.439, eps=0.005936741
Epoch 188: f=   661.414, eps=0.006233578
Epoch 189: f=   661.388, eps=0.006545257
Epoch 190: f=   661.361, eps=0.006872520
Epoch 191: f=   661.333, eps=0.007216146
Epoch 192: f=   661.303, eps=0.007576953
Epoch 193: f=   661.272, eps=0.007955801
Epoch 194: f=   661.240, eps=0.008353591
Epoch 195: f=   661.206, eps=0.008771270
Epoch 196: f=   661.170, eps=0.009209834
Epoch 197: f=   661.133, eps=0.009670325
```

```
Epoch 198: f=   661.097, eps=0.010153842
Epoch 199: f=   661.093, eps=0.010661534
Epoch 200: f=   661.463, eps=0.005330767
Epoch 201: f=   661.555, eps=0.002665383
Epoch 202: f=   660.978, eps=0.002798653
Epoch 203: f=   660.966, eps=0.002938585
Epoch 204: f=   660.955, eps=0.003085514
Epoch 205: f=   660.942, eps=0.003239790
Epoch 206: f=   660.929, eps=0.003401780
Epoch 207: f=   660.916, eps=0.003571869
Epoch 208: f=   660.902, eps=0.003750462
Epoch 209: f=   660.887, eps=0.003937985
Epoch 210: f=   660.871, eps=0.004134885
Epoch 211: f=   660.855, eps=0.004341629
Epoch 212: f=   660.837, eps=0.004558710
Epoch 213: f=   660.819, eps=0.004786646
Epoch 214: f=   660.801, eps=0.005025978
Epoch 215: f=   660.781, eps=0.005277277
Epoch 216: f=   660.760, eps=0.005541141
Epoch 217: f=   660.738, eps=0.005818198
Epoch 218: f=   660.715, eps=0.006109108
Epoch 219: f=   660.691, eps=0.006414563
Epoch 220: f=   660.666, eps=0.006735291
Epoch 221: f=   660.640, eps=0.007072056
Epoch 222: f=   660.612, eps=0.007425659
Epoch 223: f=   660.583, eps=0.007796941
Epoch 224: f=   660.553, eps=0.008186788
Epoch 225: f=   660.521, eps=0.008596128
Epoch 226: f=   660.488, eps=0.009025934
Epoch 227: f=   660.453, eps=0.009477231
Epoch 228: f=   660.417, eps=0.009951093
Epoch 229: f=   660.379, eps=0.010448647
Epoch 230: f=   660.344, eps=0.010971080
Epoch 231: f=   660.362, eps=0.005485540
Epoch 232: f=   660.377, eps=0.002742770
Epoch 233: f=   660.267, eps=0.002879908
Epoch 234: f=   660.254, eps=0.003023904
Epoch 235: f=   660.243, eps=0.003175099
Epoch 236: f=   660.231, eps=0.003333854
Epoch 237: f=   660.218, eps=0.003500547
Epoch 238: f=   660.205, eps=0.003675574
Epoch 239: f=   660.191, eps=0.003859353
Epoch 240: f=   660.176, eps=0.004052320
Epoch 241: f=   660.161, eps=0.004254936
Epoch 242: f=   660.145, eps=0.004467683
Epoch 243: f=   660.128, eps=0.004691067
Epoch 244: f=   660.111, eps=0.004925621
Epoch 245: f=   660.092, eps=0.005171902
```

```
Epoch 246: f=   660.073, eps=0.005430497
Epoch 247: f=   660.052, eps=0.005702022
Epoch 248: f=   660.031, eps=0.005987123
Epoch 249: f=   660.009, eps=0.006286479
Epoch 250: f=   659.985, eps=0.006600803
Epoch 251: f=   659.961, eps=0.006930843
Epoch 252: f=   659.935, eps=0.007277385
Epoch 253: f=   659.908, eps=0.007641254
Epoch 254: f=   659.880, eps=0.008023317
Epoch 255: f=   659.850, eps=0.008424483
Epoch 256: f=   659.819, eps=0.008845707
Epoch 257: f=   659.787, eps=0.009287992
Epoch 258: f=   659.754, eps=0.009752392
Epoch 259: f=   659.737, eps=0.010240012
Epoch 260: f=   659.888, eps=0.005120006
Epoch 261: f=   659.906, eps=0.002560003
Epoch 262: f=   659.651, eps=0.002688003
Epoch 263: f=   659.641, eps=0.002822403
Epoch 264: f=   659.631, eps=0.002963523
Epoch 265: f=   659.620, eps=0.003111700
Epoch 266: f=   659.609, eps=0.003267285
Epoch 267: f=   659.597, eps=0.003430649
Epoch 268: f=   659.585, eps=0.003602181
Epoch 269: f=   659.572, eps=0.003782290
Epoch 270: f=   659.558, eps=0.003971405
Epoch 271: f=   659.543, eps=0.004169975
Epoch 272: f=   659.528, eps=0.004378474
Epoch 273: f=   659.513, eps=0.004597397
Epoch 274: f=   659.496, eps=0.004827267
Epoch 275: f=   659.479, eps=0.005068631
Epoch 276: f=   659.460, eps=0.005322062
Epoch 277: f=   659.441, eps=0.005588165
Epoch 278: f=   659.421, eps=0.005867574
Epoch 279: f=   659.400, eps=0.006160952
Epoch 280: f=   659.378, eps=0.006469000
Epoch 281: f=   659.355, eps=0.006792450
Epoch 282: f=   659.331, eps=0.007132072
Epoch 283: f=   659.305, eps=0.007488676
Epoch 284: f=   659.279, eps=0.007863110
Epoch 285: f=   659.251, eps=0.008256265
Epoch 286: f=   659.222, eps=0.008669078
Epoch 287: f=   659.191, eps=0.009102532
Epoch 288: f=   659.159, eps=0.009557659
Epoch 289: f=   659.125, eps=0.010035542
Epoch 290: f=   659.090, eps=0.010537319
Epoch 291: f=   659.053, eps=0.011064185
Epoch 292: f=   659.016, eps=0.011617394
Epoch 293: f=   658.992, eps=0.012198264
```

```
Epoch 294: f=   659.226, eps=0.006099132
Epoch 295: f=   659.526, eps=0.003049566
Epoch 296: f=   658.916, eps=0.003202044
Epoch 297: f=   658.891, eps=0.003362147
Epoch 298: f=   658.878, eps=0.003530254
Epoch 299: f=   658.865, eps=0.003706767
Epoch 300: f=   658.852, eps=0.003892105
Epoch 301: f=   658.839, eps=0.004086710
Epoch 302: f=   658.825, eps=0.004291046
Epoch 303: f=   658.810, eps=0.004505598
Epoch 304: f=   658.795, eps=0.004730878
Epoch 305: f=   658.778, eps=0.004967422
Epoch 306: f=   658.761, eps=0.005215793
Epoch 307: f=   658.743, eps=0.005476582
Epoch 308: f=   658.725, eps=0.005750412
Epoch 309: f=   658.705, eps=0.006037932
Epoch 310: f=   658.684, eps=0.006339829
Epoch 311: f=   658.663, eps=0.006656820
Epoch 312: f=   658.640, eps=0.006989661
Epoch 313: f=   658.617, eps=0.007339144
Epoch 314: f=   658.593, eps=0.007706101
Epoch 315: f=   658.573, eps=0.008091406
Epoch 316: f=   658.582, eps=0.004045703
Epoch 317: f=   658.544, eps=0.004247988
Epoch 318: f=   658.521, eps=0.004460388
Epoch 319: f=   658.503, eps=0.004683407
Epoch 320: f=   658.486, eps=0.004917578
Epoch 321: f=   658.470, eps=0.005163456
Epoch 322: f=   658.455, eps=0.005421629
Epoch 323: f=   658.443, eps=0.005692711
Epoch 324: f=   658.436, eps=0.005977346
Epoch 325: f=   658.450, eps=0.002988673
Epoch 326: f=   658.381, eps=0.003138107
Epoch 327: f=   658.368, eps=0.003295012
Epoch 328: f=   658.356, eps=0.003459763
Epoch 329: f=   658.345, eps=0.003632751
Epoch 330: f=   658.333, eps=0.003814388
Epoch 331: f=   658.320, eps=0.004005108
Epoch 332: f=   658.307, eps=0.004205363
Epoch 333: f=   658.293, eps=0.004415631
Epoch 334: f=   658.278, eps=0.004636413
Epoch 335: f=   658.263, eps=0.004868234
Epoch 336: f=   658.247, eps=0.005111645
Epoch 337: f=   658.230, eps=0.005367228
Epoch 338: f=   658.212, eps=0.005635589
Epoch 339: f=   658.193, eps=0.005917368
Epoch 340: f=   658.174, eps=0.006213237
Epoch 341: f=   658.153, eps=0.006523899
```

```
Epoch 342: f=   658.132, eps=0.006850094
Epoch 343: f=   658.109, eps=0.007192598
Epoch 344: f=   658.086, eps=0.007552228
Epoch 345: f=   658.061, eps=0.007929840
Epoch 346: f=   658.036, eps=0.008326332
Epoch 347: f=   658.017, eps=0.008742648
Epoch 348: f=   658.040, eps=0.004371324
Epoch 349: f=   658.004, eps=0.004589890
Epoch 350: f=   657.981, eps=0.004819385
Epoch 351: f=   657.965, eps=0.005060354
Epoch 352: f=   657.954, eps=0.005313372
Epoch 353: f=   657.953, eps=0.005579040
Epoch 354: f=   657.969, eps=0.002789520
Epoch 355: f=   657.876, eps=0.002928996
Epoch 356: f=   657.864, eps=0.003075446
Epoch 357: f=   657.854, eps=0.003229218
Epoch 358: f=   657.844, eps=0.003390679
Epoch 359: f=   657.833, eps=0.003560213
Epoch 360: f=   657.821, eps=0.003738224
Epoch 361: f=   657.809, eps=0.003925135
Epoch 362: f=   657.797, eps=0.004121392
Epoch 363: f=   657.783, eps=0.004327461
Epoch 364: f=   657.770, eps=0.004543834
Epoch 365: f=   657.755, eps=0.004771026
Epoch 366: f=   657.740, eps=0.005009577
Epoch 367: f=   657.724, eps=0.005260056
Epoch 368: f=   657.707, eps=0.005523059
Epoch 369: f=   657.689, eps=0.005799212
Epoch 370: f=   657.671, eps=0.006089173
Epoch 371: f=   657.651, eps=0.006393631
Epoch 372: f=   657.631, eps=0.006713313
Epoch 373: f=   657.609, eps=0.007048978
Epoch 374: f=   657.587, eps=0.007401427
Epoch 375: f=   657.564, eps=0.007771499
Epoch 376: f=   657.539, eps=0.008160074
Epoch 377: f=   657.513, eps=0.008568077
Epoch 378: f=   657.486, eps=0.008996481
Epoch 379: f=   657.460, eps=0.009446305
Epoch 380: f=   657.445, eps=0.009918621
Epoch 381: f=   657.554, eps=0.004959310
Epoch 382: f=   657.540, eps=0.005207276
Epoch 383: f=   657.567, eps=0.002603638
Epoch 384: f=   657.357, eps=0.002733820
Epoch 385: f=   657.348, eps=0.002870511
Epoch 386: f=   657.339, eps=0.003014036
Epoch 387: f=   657.330, eps=0.003164738
Epoch 388: f=   657.320, eps=0.003322975
Epoch 389: f=   657.310, eps=0.003489124
```

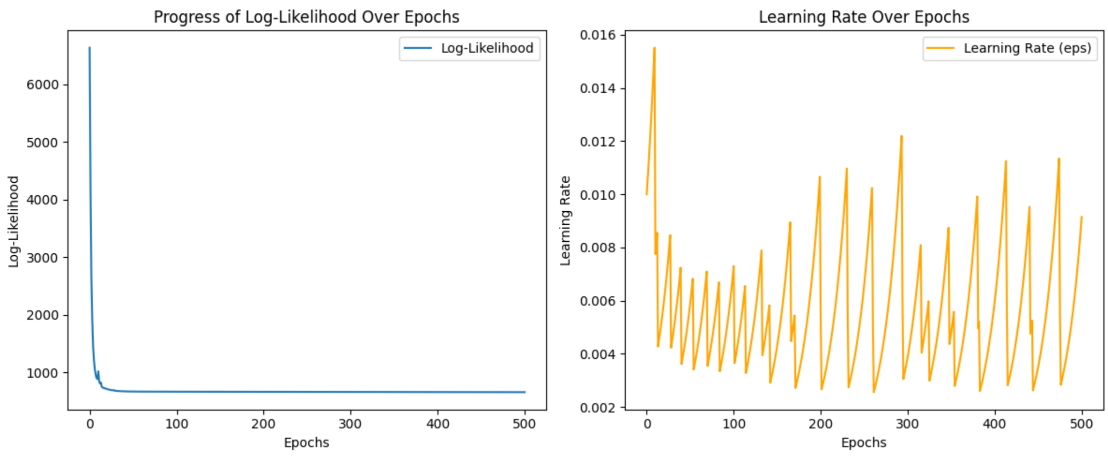
```
Epoch 390: f=   657.299, eps=0.003663580
Epoch 391: f=   657.287, eps=0.003846759
Epoch 392: f=   657.275, eps=0.004039097
Epoch 393: f=   657.263, eps=0.004241052
Epoch 394: f=   657.250, eps=0.004453104
Epoch 395: f=   657.236, eps=0.004675760
Epoch 396: f=   657.221, eps=0.004909548
Epoch 397: f=   657.206, eps=0.005155025
Epoch 398: f=   657.190, eps=0.005412776
Epoch 399: f=   657.173, eps=0.005683415
Epoch 400: f=   657.156, eps=0.005967586
Epoch 401: f=   657.138, eps=0.006265965
Epoch 402: f=   657.118, eps=0.006579263
Epoch 403: f=   657.098, eps=0.006908226
Epoch 404: f=   657.077, eps=0.007253638
Epoch 405: f=   657.054, eps=0.007616320
Epoch 406: f=   657.031, eps=0.007997136
Epoch 407: f=   657.007, eps=0.008396992
Epoch 408: f=   656.981, eps=0.008816842
Epoch 409: f=   656.954, eps=0.009257684
Epoch 410: f=   656.926, eps=0.009720568
Epoch 411: f=   656.896, eps=0.010206597
Epoch 412: f=   656.866, eps=0.010716927
Epoch 413: f=   656.838, eps=0.011252773
Epoch 414: f=   656.871, eps=0.005626387
Epoch 415: f=   656.908, eps=0.002813193
Epoch 416: f=   656.776, eps=0.002953853
Epoch 417: f=   656.765, eps=0.003101546
Epoch 418: f=   656.755, eps=0.003256623
Epoch 419: f=   656.745, eps=0.003419454
Epoch 420: f=   656.735, eps=0.003590427
Epoch 421: f=   656.724, eps=0.003769948
Epoch 422: f=   656.713, eps=0.003958445
Epoch 423: f=   656.701, eps=0.004156368
Epoch 424: f=   656.689, eps=0.004364186
Epoch 425: f=   656.676, eps=0.004582395
Epoch 426: f=   656.662, eps=0.004811515
Epoch 427: f=   656.648, eps=0.005052091
Epoch 428: f=   656.632, eps=0.005304695
Epoch 429: f=   656.617, eps=0.005569930
Epoch 430: f=   656.600, eps=0.005848427
Epoch 431: f=   656.583, eps=0.006140848
Epoch 432: f=   656.564, eps=0.006447890
Epoch 433: f=   656.545, eps=0.006770285
Epoch 434: f=   656.525, eps=0.007108799
Epoch 435: f=   656.504, eps=0.007464239
Epoch 436: f=   656.482, eps=0.007837451
Epoch 437: f=   656.459, eps=0.008229324
```

```
Epoch 438: f=   656.435, eps=0.008640790
Epoch 439: f=   656.410, eps=0.009072829
Epoch 440: f=   656.388, eps=0.009526471
Epoch 441: f=   656.406, eps=0.004763235
Epoch 442: f=   656.387, eps=0.005001397
Epoch 443: f=   656.379, eps=0.005251467
Epoch 444: f=   656.381, eps=0.002625734
Epoch 445: f=   656.303, eps=0.002757020
Epoch 446: f=   656.295, eps=0.002894871
Epoch 447: f=   656.286, eps=0.003039615
Epoch 448: f=   656.277, eps=0.003191596
Epoch 449: f=   656.268, eps=0.003351175
Epoch 450: f=   656.258, eps=0.003518734
Epoch 451: f=   656.248, eps=0.003694671
Epoch 452: f=   656.237, eps=0.003879404
Epoch 453: f=   656.226, eps=0.004073375
Epoch 454: f=   656.214, eps=0.004277043
Epoch 455: f=   656.202, eps=0.004490895
Epoch 456: f=   656.189, eps=0.004715440
Epoch 457: f=   656.175, eps=0.004951212
Epoch 458: f=   656.161, eps=0.005198773
Epoch 459: f=   656.145, eps=0.005458711
Epoch 460: f=   656.130, eps=0.005731647
Epoch 461: f=   656.113, eps=0.006018229
Epoch 462: f=   656.096, eps=0.006319141
Epoch 463: f=   656.077, eps=0.006635098
Epoch 464: f=   656.058, eps=0.006966853
Epoch 465: f=   656.038, eps=0.007315195
Epoch 466: f=   656.017, eps=0.007680955
Epoch 467: f=   655.995, eps=0.008065003
Epoch 468: f=   655.972, eps=0.008468253
Epoch 469: f=   655.948, eps=0.008891666
Epoch 470: f=   655.923, eps=0.009336249
Epoch 471: f=   655.896, eps=0.009803061
Epoch 472: f=   655.868, eps=0.010293215
Epoch 473: f=   655.841, eps=0.010807875
Epoch 474: f=   655.835, eps=0.011348269
Epoch 475: f=   656.135, eps=0.005674135
Epoch 476: f=   656.301, eps=0.002837067
Epoch 477: f=   655.760, eps=0.002978921
Epoch 478: f=   655.744, eps=0.003127867
Epoch 479: f=   655.735, eps=0.003284260
Epoch 480: f=   655.725, eps=0.003448473
Epoch 481: f=   655.716, eps=0.003620897
Epoch 482: f=   655.705, eps=0.003801941
Epoch 483: f=   655.695, eps=0.003992039
Epoch 484: f=   655.684, eps=0.004191640
Epoch 485: f=   655.672, eps=0.004401222
```

```
Epoch 486: f=   655.659, eps=0.004621284
Epoch 487: f=   655.646, eps=0.004852348
Epoch 488: f=   655.633, eps=0.005094965
Epoch 489: f=   655.619, eps=0.005349713
Epoch 490: f=   655.604, eps=0.005617199
Epoch 491: f=   655.588, eps=0.005898059
Epoch 492: f=   655.571, eps=0.006192962
Epoch 493: f=   655.554, eps=0.006502610
Epoch 494: f=   655.536, eps=0.006827741
Epoch 495: f=   655.517, eps=0.007169128
Epoch 496: f=   655.497, eps=0.007527584
Epoch 497: f=   655.476, eps=0.007903963
Epoch 498: f=   655.454, eps=0.008299161
Epoch 499: f=   655.432, eps=0.008714119
Result after 500 epochs: f=655.413496469942
```

[129]:

```
# look at how gradient descent made progess
import matplotlib.pyplot as plt

# Plot the log-likelihood over epochs
plt.figure(figsize=(12, 5))

# Plot log-likelihood values
plt.subplot(1, 2, 1)
plt.plot(vz_gd, label="Log-Likelihood")
plt.xlabel("Epochs")
plt.ylabel("Log-Likelihood")
plt.title("Progress of Log-Likelihood Over Epochs")
plt.legend()

# Plot learning rate (eps) over epochs
plt.subplot(1, 2, 2)
plt.plot(ez_gd, label="Learning Rate (eps)", color="orange")
plt.xlabel("Epochs")
plt.ylabel("Learning Rate")
plt.title("Learning Rate Over Epochs")
plt.legend()

plt.tight_layout()
plt.show()
```



3.4 2d Stochastic gradient descent

[130]:
```python
import numpy as np

def sgdepoch(y, X, w, eps):
    """Run one SGD epoch and return the updated weight vector. """
    # Run N stochastic gradient steps (without replacement). Do not rescale each
    # step by factor N (i.e., proceed differently than in the lecture slides).

    # Shuffle the data points (without replacement)
    indices = np.random.permutation(len(y))

    # Perform a gradient update for each data point
    for i in indices:
        xi = X[i]
        yi = y[i]
        pi = sigma(xi @ w)
        ei = yi - pi
        gradient = ei * xi
        w = w + eps * gradient

    return w
```

[131]:
```python
# when you run this multiple times, with 50% probability you should get the
# following result (there is one other result which is very close):
# array([ -3.43689655e+02,  -1.71161311e+02,  -5.71093536e+02,
#         -5.16478220e+01,   4.66294348e+02,  -3.71589878e+02,
#          5.21493183e+02,   1.25699230e+03,   8.33804130e+02,
#          5.63185399e+02,   1.32761302e+03,  -2.64104011e+02,
#          7.10693307e+02,  -1.75497331e+02,  -1.94174427e+02,
#          1.11641507e+02,  -3.30817509e+02,  -3.46754913e+02,
```

```
#          8.48722111e+02,  -1.89136304e+02,  -4.25693844e+02,
#         -1.23084189e+02,  -2.95894797e+02,  -2.35789333e+02,
#         -3.38695243e+02,  -3.05642830e+02,  -2.28975383e+02,
#         -2.38075137e+02,  -1.66702530e+02,  -2.27341599e+02,
#         -1.77575620e+02,  -1.49093855e+02,  -1.70028859e+02,
#         -1.50243833e+02,  -1.82986008e+02,  -2.41143708e+02,
#         -3.31047159e+02,  -5.79991185e+01,  -1.98477863e+02,
#         -1.91264948e+02,  -1.17371919e+02,  -1.66953779e+02,
#         -2.01472565e+02,  -1.23330949e+02,  -3.00857740e+02,
#         -1.95853348e+02,  -7.44868073e+01,  -1.11172370e+02,
#         -1.57618226e+02,  -1.25729512e+00,  -1.45536466e+02,
#         -1.43362438e+02,  -3.00429708e+02,  -9.84391082e+01,
#         -4.54152047e+01,  -5.26492232e+01,  -1.45175427e+02])
sgdepoch(y[1:3], Xz[1:3, :], np.linspace(-5, 5, D), 1000)
```

[131]:
```
array([-3.43689655e+02, -1.71161311e+02, -5.71093536e+02, -5.16478220e+01,
        4.66294348e+02, -3.71589878e+02,  5.21493183e+02,  1.25699230e+03,
        8.33804130e+02,  5.63185399e+02,  1.32761302e+03, -2.64104011e+02,
        7.10693307e+02, -1.75497331e+02, -1.94174427e+02,  1.11641507e+02,
       -3.30817509e+02, -3.46754913e+02,  8.48722111e+02, -1.89136304e+02,
       -4.25693844e+02, -1.23084189e+02, -2.95894797e+02, -2.35789333e+02,
       -3.38695243e+02, -3.05642830e+02, -2.28975383e+02, -2.38075137e+02,
       -1.66702530e+02, -2.27341599e+02, -1.77575620e+02, -1.49093855e+02,
       -1.70028859e+02, -1.50243833e+02, -1.82986008e+02, -2.41143708e+02,
       -3.31047159e+02, -5.79991185e+01, -1.98477863e+02, -1.91264948e+02,
       -1.17371919e+02, -1.66953779e+02, -2.01472565e+02, -1.23330949e+02,
       -3.00857740e+02, -1.95853348e+02, -7.44868073e+01, -1.11172370e+02,
       -1.57618226e+02, -1.25729512e+00, -1.45536466e+02, -1.43362438e+02,
       -3.00429708e+02, -9.84391082e+01, -4.54152047e+01, -5.26492232e+01,
       -1.45175427e+02])
```

[132]:
```
# define the objective and update function for one gradient-descent epoch for
# fitting an MLE estimate of logistic regression with stochastic gradient␣
↪descent
# (should return a tuple of two functions; see optimize)
def sgd(y, X):
    def objective(w):
        return -l(y, X, w)

    def update(w, eps):
        return sgdepoch(y, X, w, eps)

    return (objective, update)
```

[133]:
```
# with 50% probability, you should get:
# [40.864973045695081,
#  array([ -3.43689655e+02,  -1.71161311e+02,  -5.71093536e+02,
```

```
#           -5.16478220e+01,   4.66294348e+02,  -3.71589878e+02,
#            5.21493183e+02,   1.25699230e+03,   8.33804130e+02,
#            5.63185399e+02,   1.32761302e+03,  -2.64104011e+02,
#            7.10693307e+02,  -1.75497331e+02,  -1.94174427e+02,
#            1.11641507e+02,  -3.30817509e+02,  -3.46754913e+02,
#            8.48722111e+02,  -1.89136304e+02,  -4.25693844e+02,
#           -1.23084189e+02,  -2.95894797e+02,  -2.35789333e+02,
#           -3.38695243e+02,  -3.05642830e+02,  -2.28975383e+02,
#           -2.38075137e+02,  -1.66702530e+02,  -2.27341599e+02,
#           -1.77575620e+02,  -1.49093855e+02,  -1.70028859e+02,
#           -1.50243833e+02,  -1.82986008e+02,  -2.41143708e+02,
#           -3.31047159e+02,  -5.79991185e+01,  -1.98477863e+02,
#           -1.91264948e+02,  -1.17371919e+02,  -1.66953779e+02,
#           -2.01472565e+02,  -1.23330949e+02,  -3.00857740e+02,
#           -1.95853348e+02,  -7.44868073e+01,  -1.11172370e+02,
#           -1.57618226e+02,  -1.25729512e+00,  -1.45536466e+02,
#           -1.43362438e+02,  -3.00429708e+02,  -9.84391082e+01,
#           -4.54152047e+01,  -5.26492232e+01,  -1.45175427e+02])]
f, update = sgd(y[1:3], Xz[1:3, :])
[f(np.linspace(-5, 5, D)), update(np.linspace(-5, 5, D), 1000)]
```

[133]:
```
[40.86497304569509,
 array([-3.43689655e+02, -1.71161311e+02, -5.71093536e+02, -5.16478220e+01,
         4.66294348e+02, -3.71589878e+02,  5.21493183e+02,  1.25699230e+03,
         8.33804130e+02,  5.63185399e+02,  1.32761302e+03, -2.64104011e+02,
         7.10693307e+02, -1.75497331e+02, -1.94174427e+02,  1.11641507e+02,
        -3.30817509e+02, -3.46754913e+02,  8.48722111e+02, -1.89136304e+02,
        -4.25693844e+02, -1.23084189e+02, -2.95894797e+02, -2.35789333e+02,
        -3.38695243e+02, -3.05642830e+02, -2.28975383e+02, -2.38075137e+02,
        -1.66702530e+02, -2.27341599e+02, -1.77575620e+02, -1.49093855e+02,
        -1.70028859e+02, -1.50243833e+02, -1.82986008e+02, -2.41143708e+02,
        -3.31047159e+02, -5.79991185e+01, -1.98477863e+02, -1.91264948e+02,
        -1.17371919e+02, -1.66953779e+02, -2.01472565e+02, -1.23330949e+02,
        -3.00857740e+02, -1.95853348e+02, -7.44868073e+01, -1.11172370e+02,
        -1.57618226e+02, -1.25729512e+00, -1.45536466e+02, -1.43362438e+02,
        -3.00429708e+02, -9.84391082e+01, -4.54152047e+01, -5.26492232e+01,
        -1.45175427e+02])]
```

[134]:
```
# you can run stochastic gradient descent!
wz_sgd, vz_sgd, ez_sgd = optimize(sgd(y, Xz), w0, nepochs=500)
```

```
Epoch   0: f=  6636.208, eps=0.010000000
Epoch   1: f=   958.654, eps=0.010500000
Epoch   2: f=   786.651, eps=0.011025000
Epoch   3: f=   738.739, eps=0.011576250
Epoch   4: f=   718.166, eps=0.012155063
Epoch   5: f=   709.413, eps=0.012762816
```

```
Epoch   6: f=   696.048, eps=0.013400956
Epoch   7: f=   701.674, eps=0.006700478
Epoch   8: f=   686.406, eps=0.007035502
Epoch   9: f=   683.692, eps=0.007387277
Epoch  10: f=   684.500, eps=0.003693639
Epoch  11: f=   679.969, eps=0.003878321
Epoch  12: f=   679.230, eps=0.004072237
Epoch  13: f=   678.181, eps=0.004275848
Epoch  14: f=   677.570, eps=0.004489641
Epoch  15: f=   676.763, eps=0.004714123
Epoch  16: f=   675.966, eps=0.004949829
Epoch  17: f=   676.625, eps=0.002474914
Epoch  18: f=   675.185, eps=0.002598660
Epoch  19: f=   674.531, eps=0.002728593
Epoch  20: f=   674.095, eps=0.002865023
Epoch  21: f=   673.692, eps=0.003008274
Epoch  22: f=   673.359, eps=0.003158688
Epoch  23: f=   673.075, eps=0.003316622
Epoch  24: f=   672.914, eps=0.003482453
Epoch  25: f=   672.549, eps=0.003656576
Epoch  26: f=   672.208, eps=0.003839405
Epoch  27: f=   672.128, eps=0.004031375
Epoch  28: f=   671.943, eps=0.004232944
Epoch  29: f=   671.364, eps=0.004444591
Epoch  30: f=   671.437, eps=0.002222295
Epoch  31: f=   670.780, eps=0.002333410
Epoch  32: f=   670.619, eps=0.002450081
Epoch  33: f=   670.405, eps=0.002572585
Epoch  34: f=   670.230, eps=0.002701214
Epoch  35: f=   670.065, eps=0.002836275
Epoch  36: f=   669.883, eps=0.002978088
Epoch  37: f=   669.804, eps=0.003126993
Epoch  38: f=   669.708, eps=0.003283342
Epoch  39: f=   669.592, eps=0.003447510
Epoch  40: f=   669.531, eps=0.003619885
Epoch  41: f=   669.408, eps=0.003800879
Epoch  42: f=   669.209, eps=0.003990923
Epoch  43: f=   669.264, eps=0.001995462
Epoch  44: f=   668.761, eps=0.002095235
Epoch  45: f=   668.598, eps=0.002199996
Epoch  46: f=   668.608, eps=0.001099998
Epoch  47: f=   668.502, eps=0.001154998
Epoch  48: f=   668.400, eps=0.001212748
Epoch  49: f=   668.324, eps=0.001273385
Epoch  50: f=   668.278, eps=0.001337055
Epoch  51: f=   668.235, eps=0.001403907
Epoch  52: f=   668.189, eps=0.001474103
Epoch  53: f=   668.134, eps=0.001547808
```

```
Epoch  54: f=   668.080, eps=0.001625198
Epoch  55: f=   668.035, eps=0.001706458
Epoch  56: f=   667.966, eps=0.001791781
Epoch  57: f=   667.927, eps=0.001881370
Epoch  58: f=   667.878, eps=0.001975439
Epoch  59: f=   667.813, eps=0.002074211
Epoch  60: f=   667.757, eps=0.002177921
Epoch  61: f=   667.722, eps=0.002286817
Epoch  62: f=   667.667, eps=0.002401158
Epoch  63: f=   667.613, eps=0.002521216
Epoch  64: f=   667.515, eps=0.002647277
Epoch  65: f=   667.483, eps=0.002779641
Epoch  66: f=   667.370, eps=0.002918623
Epoch  67: f=   667.319, eps=0.003064554
Epoch  68: f=   667.369, eps=0.001532277
Epoch  69: f=   667.264, eps=0.001608891
Epoch  70: f=   667.200, eps=0.001689335
Epoch  71: f=   667.172, eps=0.001773802
Epoch  72: f=   667.112, eps=0.001862492
Epoch  73: f=   667.006, eps=0.001955617
Epoch  74: f=   667.045, eps=0.000977808
Epoch  75: f=   666.973, eps=0.001026699
Epoch  76: f=   666.911, eps=0.001078034
Epoch  77: f=   666.884, eps=0.001131935
Epoch  78: f=   666.844, eps=0.001188532
Epoch  79: f=   666.823, eps=0.001247959
Epoch  80: f=   666.794, eps=0.001310357
Epoch  81: f=   666.766, eps=0.001375875
Epoch  82: f=   666.745, eps=0.001444668
Epoch  83: f=   666.708, eps=0.001516902
Epoch  84: f=   666.707, eps=0.001592747
Epoch  85: f=   666.649, eps=0.001672384
Epoch  86: f=   666.632, eps=0.001756003
Epoch  87: f=   666.610, eps=0.001843804
Epoch  88: f=   666.579, eps=0.001935994
Epoch  89: f=   666.532, eps=0.002032793
Epoch  90: f=   666.482, eps=0.002134433
Epoch  91: f=   666.461, eps=0.002241155
Epoch  92: f=   666.440, eps=0.002353213
Epoch  93: f=   666.365, eps=0.002470873
Epoch  94: f=   666.433, eps=0.001235437
Epoch  95: f=   666.342, eps=0.001297208
Epoch  96: f=   666.294, eps=0.001362069
Epoch  97: f=   666.259, eps=0.001430172
Epoch  98: f=   666.240, eps=0.001501681
Epoch  99: f=   666.209, eps=0.001576765
Epoch 100: f=   666.185, eps=0.001655603
Epoch 101: f=   666.153, eps=0.001738383
```

```
Epoch 102: f=   666.153, eps=0.001825303
Epoch 103: f=   666.166, eps=0.000912651
Epoch 104: f=   666.137, eps=0.000958284
Epoch 105: f=   666.104, eps=0.001006198
Epoch 106: f=   666.091, eps=0.001056508
Epoch 107: f=   666.047, eps=0.001109333
Epoch 108: f=   666.025, eps=0.001164800
Epoch 109: f=   666.006, eps=0.001223040
Epoch 110: f=   665.992, eps=0.001284192
Epoch 111: f=   665.967, eps=0.001348402
Epoch 112: f=   665.966, eps=0.001415822
Epoch 113: f=   665.960, eps=0.001486613
Epoch 114: f=   665.916, eps=0.001560943
Epoch 115: f=   665.935, eps=0.000780472
Epoch 116: f=   665.913, eps=0.000819495
Epoch 117: f=   665.879, eps=0.000860470
Epoch 118: f=   665.865, eps=0.000903494
Epoch 119: f=   665.855, eps=0.000948668
Epoch 120: f=   665.834, eps=0.000996102
Epoch 121: f=   665.824, eps=0.001045907
Epoch 122: f=   665.811, eps=0.001098202
Epoch 123: f=   665.786, eps=0.001153112
Epoch 124: f=   665.777, eps=0.001210768
Epoch 125: f=   665.778, eps=0.000605384
Epoch 126: f=   665.761, eps=0.000635653
Epoch 127: f=   665.745, eps=0.000667436
Epoch 128: f=   665.737, eps=0.000700808
Epoch 129: f=   665.725, eps=0.000735848
Epoch 130: f=   665.716, eps=0.000772640
Epoch 131: f=   665.704, eps=0.000811272
Epoch 132: f=   665.700, eps=0.000851836
Epoch 133: f=   665.687, eps=0.000894428
Epoch 134: f=   665.676, eps=0.000939149
Epoch 135: f=   665.662, eps=0.000986107
Epoch 136: f=   665.651, eps=0.001035412
Epoch 137: f=   665.640, eps=0.001087182
Epoch 138: f=   665.630, eps=0.001141542
Epoch 139: f=   665.614, eps=0.001198619
Epoch 140: f=   665.602, eps=0.001258550
Epoch 141: f=   665.586, eps=0.001321477
Epoch 142: f=   665.573, eps=0.001387551
Epoch 143: f=   665.570, eps=0.001456928
Epoch 144: f=   665.538, eps=0.001529775
Epoch 145: f=   665.519, eps=0.001606264
Epoch 146: f=   665.536, eps=0.000803132
Epoch 147: f=   665.500, eps=0.000843288
Epoch 148: f=   665.486, eps=0.000885453
Epoch 149: f=   665.471, eps=0.000929725
```

```
Epoch 150: f=   665.461, eps=0.000976212
Epoch 151: f=   665.451, eps=0.001025022
Epoch 152: f=   665.439, eps=0.001076273
Epoch 153: f=   665.432, eps=0.001130087
Epoch 154: f=   665.418, eps=0.001186591
Epoch 155: f=   665.420, eps=0.000593296
Epoch 156: f=   665.403, eps=0.000622961
Epoch 157: f=   665.393, eps=0.000654109
Epoch 158: f=   665.383, eps=0.000686814
Epoch 159: f=   665.376, eps=0.000721155
Epoch 160: f=   665.366, eps=0.000757212
Epoch 161: f=   665.357, eps=0.000795073
Epoch 162: f=   665.350, eps=0.000834827
Epoch 163: f=   665.344, eps=0.000876568
Epoch 164: f=   665.333, eps=0.000920396
Epoch 165: f=   665.325, eps=0.000966416
Epoch 166: f=   665.316, eps=0.001014737
Epoch 167: f=   665.313, eps=0.001065474
Epoch 168: f=   665.296, eps=0.001118748
Epoch 169: f=   665.280, eps=0.001174685
Epoch 170: f=   665.267, eps=0.001233419
Epoch 171: f=   665.265, eps=0.001295090
Epoch 172: f=   665.251, eps=0.001359845
Epoch 173: f=   665.235, eps=0.001427837
Epoch 174: f=   665.250, eps=0.000713918
Epoch 175: f=   665.225, eps=0.000749614
Epoch 176: f=   665.214, eps=0.000787095
Epoch 177: f=   665.202, eps=0.000826450
Epoch 178: f=   665.192, eps=0.000867772
Epoch 179: f=   665.190, eps=0.000911161
Epoch 180: f=   665.176, eps=0.000956719
Epoch 181: f=   665.166, eps=0.001004555
Epoch 182: f=   665.151, eps=0.001054783
Epoch 183: f=   665.146, eps=0.001107522
Epoch 184: f=   665.138, eps=0.001162898
Epoch 185: f=   665.121, eps=0.001221043
Epoch 186: f=   665.106, eps=0.001282095
Epoch 187: f=   665.099, eps=0.001346200
Epoch 188: f=   665.096, eps=0.001413510
Epoch 189: f=   665.087, eps=0.001484185
Epoch 190: f=   665.082, eps=0.001558394
Epoch 191: f=   665.062, eps=0.001636314
Epoch 192: f=   665.038, eps=0.001718130
Epoch 193: f=   665.037, eps=0.001804036
Epoch 194: f=   665.042, eps=0.000902018
Epoch 195: f=   665.013, eps=0.000947119
Epoch 196: f=   664.992, eps=0.000994475
Epoch 197: f=   664.982, eps=0.001044199
```

```
Epoch 198: f=   664.975, eps=0.001096409
Epoch 199: f=   664.970, eps=0.001151229
Epoch 200: f=   664.960, eps=0.001208791
Epoch 201: f=   664.939, eps=0.001269230
Epoch 202: f=   664.944, eps=0.000634615
Epoch 203: f=   664.928, eps=0.000666346
Epoch 204: f=   664.916, eps=0.000699663
Epoch 205: f=   664.908, eps=0.000734646
Epoch 206: f=   664.900, eps=0.000771379
Epoch 207: f=   664.891, eps=0.000809948
Epoch 208: f=   664.886, eps=0.000850445
Epoch 209: f=   664.876, eps=0.000892967
Epoch 210: f=   664.870, eps=0.000937616
Epoch 211: f=   664.863, eps=0.000984496
Epoch 212: f=   664.856, eps=0.001033721
Epoch 213: f=   664.847, eps=0.001085407
Epoch 214: f=   664.842, eps=0.001139678
Epoch 215: f=   664.840, eps=0.001196661
Epoch 216: f=   664.823, eps=0.001256494
Epoch 217: f=   664.812, eps=0.001319319
Epoch 218: f=   664.828, eps=0.000659660
Epoch 219: f=   664.812, eps=0.000692643
Epoch 220: f=   664.796, eps=0.000727275
Epoch 221: f=   664.785, eps=0.000763638
Epoch 222: f=   664.779, eps=0.000801820
Epoch 223: f=   664.769, eps=0.000841911
Epoch 224: f=   664.763, eps=0.000884007
Epoch 225: f=   664.757, eps=0.000928207
Epoch 226: f=   664.747, eps=0.000974618
Epoch 227: f=   664.740, eps=0.001023349
Epoch 228: f=   664.730, eps=0.001074516
Epoch 229: f=   664.721, eps=0.001128242
Epoch 230: f=   664.713, eps=0.001184654
Epoch 231: f=   664.707, eps=0.001243887
Epoch 232: f=   664.706, eps=0.001306081
Epoch 233: f=   664.685, eps=0.001371385
Epoch 234: f=   664.690, eps=0.000685692
Epoch 235: f=   664.680, eps=0.000719977
Epoch 236: f=   664.675, eps=0.000755976
Epoch 237: f=   664.666, eps=0.000793775
Epoch 238: f=   664.659, eps=0.000833463
Epoch 239: f=   664.653, eps=0.000875137
Epoch 240: f=   664.636, eps=0.000918893
Epoch 241: f=   664.634, eps=0.000964838
Epoch 242: f=   664.624, eps=0.001013080
Epoch 243: f=   664.614, eps=0.001063734
Epoch 244: f=   664.608, eps=0.001116921
Epoch 245: f=   664.598, eps=0.001172767
```

```
Epoch 246: f=   664.592, eps=0.001231405
Epoch 247: f=   664.588, eps=0.001292975
Epoch 248: f=   664.581, eps=0.001357624
Epoch 249: f=   664.579, eps=0.001425505
Epoch 250: f=   664.566, eps=0.001496781
Epoch 251: f=   664.579, eps=0.000748390
Epoch 252: f=   664.553, eps=0.000785810
Epoch 253: f=   664.543, eps=0.000825100
Epoch 254: f=   664.524, eps=0.000866355
Epoch 255: f=   664.517, eps=0.000909673
Epoch 256: f=   664.511, eps=0.000955157
Epoch 257: f=   664.504, eps=0.001002915
Epoch 258: f=   664.495, eps=0.001053060
Epoch 259: f=   664.488, eps=0.001105713
Epoch 260: f=   664.479, eps=0.001160999
Epoch 261: f=   664.480, eps=0.000580500
Epoch 262: f=   664.474, eps=0.000609525
Epoch 263: f=   664.467, eps=0.000640001
Epoch 264: f=   664.459, eps=0.000672001
Epoch 265: f=   664.453, eps=0.000705601
Epoch 266: f=   664.448, eps=0.000740881
Epoch 267: f=   664.441, eps=0.000777925
Epoch 268: f=   664.436, eps=0.000816821
Epoch 269: f=   664.430, eps=0.000857662
Epoch 270: f=   664.428, eps=0.000900545
Epoch 271: f=   664.417, eps=0.000945573
Epoch 272: f=   664.413, eps=0.000992851
Epoch 273: f=   664.404, eps=0.001042494
Epoch 274: f=   664.399, eps=0.001094618
Epoch 275: f=   664.393, eps=0.001149349
Epoch 276: f=   664.391, eps=0.001206817
Epoch 277: f=   664.388, eps=0.001267158
Epoch 278: f=   664.377, eps=0.001330516
Epoch 279: f=   664.378, eps=0.000665258
Epoch 280: f=   664.370, eps=0.000698521
Epoch 281: f=   664.359, eps=0.000733447
Epoch 282: f=   664.351, eps=0.000770119
Epoch 283: f=   664.349, eps=0.000808625
Epoch 284: f=   664.343, eps=0.000849056
Epoch 285: f=   664.332, eps=0.000891509
Epoch 286: f=   664.326, eps=0.000936084
Epoch 287: f=   664.318, eps=0.000982889
Epoch 288: f=   664.314, eps=0.001032033
Epoch 289: f=   664.310, eps=0.001083635
Epoch 290: f=   664.297, eps=0.001137817
Epoch 291: f=   664.292, eps=0.001194707
Epoch 292: f=   664.287, eps=0.001254443
Epoch 293: f=   664.275, eps=0.001317165
```

```
Epoch 294: f=   664.261, eps=0.001383023
Epoch 295: f=   664.263, eps=0.000691512
Epoch 296: f=   664.250, eps=0.000726087
Epoch 297: f=   664.244, eps=0.000762391
Epoch 298: f=   664.237, eps=0.000800511
Epoch 299: f=   664.233, eps=0.000840537
Epoch 300: f=   664.226, eps=0.000882563
Epoch 301: f=   664.219, eps=0.000926692
Epoch 302: f=   664.214, eps=0.000973026
Epoch 303: f=   664.211, eps=0.001021678
Epoch 304: f=   664.198, eps=0.001072761
Epoch 305: f=   664.193, eps=0.001126399
Epoch 306: f=   664.189, eps=0.001182719
Epoch 307: f=   664.187, eps=0.001241855
Epoch 308: f=   664.198, eps=0.000620928
Epoch 309: f=   664.177, eps=0.000651974
Epoch 310: f=   664.168, eps=0.000684573
Epoch 311: f=   664.162, eps=0.000718801
Epoch 312: f=   664.156, eps=0.000754742
Epoch 313: f=   664.147, eps=0.000792479
Epoch 314: f=   664.142, eps=0.000832103
Epoch 315: f=   664.137, eps=0.000873708
Epoch 316: f=   664.131, eps=0.000917393
Epoch 317: f=   664.127, eps=0.000963263
Epoch 318: f=   664.121, eps=0.001011426
Epoch 319: f=   664.115, eps=0.001061997
Epoch 320: f=   664.108, eps=0.001115097
Epoch 321: f=   664.116, eps=0.000557548
Epoch 322: f=   664.105, eps=0.000585426
Epoch 323: f=   664.096, eps=0.000614697
Epoch 324: f=   664.088, eps=0.000645432
Epoch 325: f=   664.083, eps=0.000677704
Epoch 326: f=   664.078, eps=0.000711589
Epoch 327: f=   664.071, eps=0.000747168
Epoch 328: f=   664.069, eps=0.000784527
Epoch 329: f=   664.064, eps=0.000823753
Epoch 330: f=   664.057, eps=0.000864941
Epoch 331: f=   664.050, eps=0.000908188
Epoch 332: f=   664.048, eps=0.000953597
Epoch 333: f=   664.044, eps=0.001001277
Epoch 334: f=   664.043, eps=0.001051341
Epoch 335: f=   664.028, eps=0.001103908
Epoch 336: f=   664.029, eps=0.000551954
Epoch 337: f=   664.022, eps=0.000579552
Epoch 338: f=   664.016, eps=0.000608529
Epoch 339: f=   664.014, eps=0.000638956
Epoch 340: f=   664.006, eps=0.000670903
Epoch 341: f=   664.003, eps=0.000704449
```

```
Epoch 342: f=   664.003, eps=0.000739671
Epoch 343: f=   664.000, eps=0.000776655
Epoch 344: f=   663.994, eps=0.000815487
Epoch 345: f=   663.984, eps=0.000856262
Epoch 346: f=   663.979, eps=0.000899075
Epoch 347: f=   663.980, eps=0.000449537
Epoch 348: f=   663.976, eps=0.000472014
Epoch 349: f=   663.971, eps=0.000495615
Epoch 350: f=   663.967, eps=0.000520396
Epoch 351: f=   663.963, eps=0.000546416
Epoch 352: f=   663.960, eps=0.000573736
Epoch 353: f=   663.953, eps=0.000602423
Epoch 354: f=   663.947, eps=0.000632544
Epoch 355: f=   663.943, eps=0.000664171
Epoch 356: f=   663.939, eps=0.000697380
Epoch 357: f=   663.934, eps=0.000732249
Epoch 358: f=   663.931, eps=0.000768861
Epoch 359: f=   663.928, eps=0.000807305
Epoch 360: f=   663.924, eps=0.000847670
Epoch 361: f=   663.918, eps=0.000890053
Epoch 362: f=   663.918, eps=0.000445027
Epoch 363: f=   663.913, eps=0.000467278
Epoch 364: f=   663.910, eps=0.000490642
Epoch 365: f=   663.907, eps=0.000515174
Epoch 366: f=   663.901, eps=0.000540933
Epoch 367: f=   663.897, eps=0.000567979
Epoch 368: f=   663.893, eps=0.000596378
Epoch 369: f=   663.890, eps=0.000626197
Epoch 370: f=   663.886, eps=0.000657507
Epoch 371: f=   663.882, eps=0.000690382
Epoch 372: f=   663.877, eps=0.000724902
Epoch 373: f=   663.877, eps=0.000761147
Epoch 374: f=   663.874, eps=0.000799204
Epoch 375: f=   663.869, eps=0.000839164
Epoch 376: f=   663.861, eps=0.000881122
Epoch 377: f=   663.855, eps=0.000925178
Epoch 378: f=   663.850, eps=0.000971437
Epoch 379: f=   663.843, eps=0.001020009
Epoch 380: f=   663.841, eps=0.001071010
Epoch 381: f=   663.841, eps=0.001124560
Epoch 382: f=   663.837, eps=0.001180788
Epoch 383: f=   663.832, eps=0.001239828
Epoch 384: f=   663.818, eps=0.001301819
Epoch 385: f=   663.807, eps=0.001366910
Epoch 386: f=   663.813, eps=0.000683455
Epoch 387: f=   663.800, eps=0.000717628
Epoch 388: f=   663.794, eps=0.000753509
Epoch 389: f=   663.788, eps=0.000791185
```

```
Epoch 390: f=   663.781, eps=0.000830744
Epoch 391: f=   663.778, eps=0.000872281
Epoch 392: f=   663.777, eps=0.000915895
Epoch 393: f=   663.766, eps=0.000961690
Epoch 394: f=   663.762, eps=0.001009774
Epoch 395: f=   663.757, eps=0.001060263
Epoch 396: f=   663.747, eps=0.001113276
Epoch 397: f=   663.745, eps=0.001168940
Epoch 398: f=   663.740, eps=0.001227387
Epoch 399: f=   663.758, eps=0.000613693
Epoch 400: f=   663.745, eps=0.000644378
Epoch 401: f=   663.735, eps=0.000676597
Epoch 402: f=   663.729, eps=0.000710427
Epoch 403: f=   663.730, eps=0.000355213
Epoch 404: f=   663.723, eps=0.000372974
Epoch 405: f=   663.718, eps=0.000391623
Epoch 406: f=   663.712, eps=0.000411204
Epoch 407: f=   663.709, eps=0.000431764
Epoch 408: f=   663.704, eps=0.000453352
Epoch 409: f=   663.701, eps=0.000476020
Epoch 410: f=   663.698, eps=0.000499821
Epoch 411: f=   663.694, eps=0.000524812
Epoch 412: f=   663.691, eps=0.000551053
Epoch 413: f=   663.686, eps=0.000578605
Epoch 414: f=   663.683, eps=0.000607536
Epoch 415: f=   663.678, eps=0.000637912
Epoch 416: f=   663.675, eps=0.000669808
Epoch 417: f=   663.669, eps=0.000703298
Epoch 418: f=   663.666, eps=0.000738463
Epoch 419: f=   663.660, eps=0.000775386
Epoch 420: f=   663.656, eps=0.000814156
Epoch 421: f=   663.651, eps=0.000854863
Epoch 422: f=   663.645, eps=0.000897607
Epoch 423: f=   663.640, eps=0.000942487
Epoch 424: f=   663.634, eps=0.000989611
Epoch 425: f=   663.632, eps=0.001039092
Epoch 426: f=   663.632, eps=0.000519546
Epoch 427: f=   663.626, eps=0.000545523
Epoch 428: f=   663.622, eps=0.000572799
Epoch 429: f=   663.615, eps=0.000601439
Epoch 430: f=   663.611, eps=0.000631511
Epoch 431: f=   663.607, eps=0.000663087
Epoch 432: f=   663.604, eps=0.000696241
Epoch 433: f=   663.600, eps=0.000731053
Epoch 434: f=   663.598, eps=0.000767606
Epoch 435: f=   663.592, eps=0.000805986
Epoch 436: f=   663.588, eps=0.000846286
Epoch 437: f=   663.582, eps=0.000888600
```

```
Epoch 438: f=   663.579, eps=0.000933030
Epoch 439: f=   663.574, eps=0.000979681
Epoch 440: f=   663.575, eps=0.000489841
Epoch 441: f=   663.570, eps=0.000514333
Epoch 442: f=   663.567, eps=0.000540049
Epoch 443: f=   663.561, eps=0.000567052
Epoch 444: f=   663.556, eps=0.000595404
Epoch 445: f=   663.553, eps=0.000625175
Epoch 446: f=   663.549, eps=0.000656433
Epoch 447: f=   663.547, eps=0.000689255
Epoch 448: f=   663.542, eps=0.000723718
Epoch 449: f=   663.537, eps=0.000759904
Epoch 450: f=   663.532, eps=0.000797899
Epoch 451: f=   663.529, eps=0.000837794
Epoch 452: f=   663.525, eps=0.000879684
Epoch 453: f=   663.523, eps=0.000923668
Epoch 454: f=   663.517, eps=0.000969851
Epoch 455: f=   663.510, eps=0.001018344
Epoch 456: f=   663.514, eps=0.000509172
Epoch 457: f=   663.510, eps=0.000534630
Epoch 458: f=   663.505, eps=0.000561362
Epoch 459: f=   663.499, eps=0.000589430
Epoch 460: f=   663.496, eps=0.000618902
Epoch 461: f=   663.490, eps=0.000649847
Epoch 462: f=   663.489, eps=0.000682339
Epoch 463: f=   663.486, eps=0.000716456
Epoch 464: f=   663.479, eps=0.000752279
Epoch 465: f=   663.476, eps=0.000789893
Epoch 466: f=   663.468, eps=0.000829387
Epoch 467: f=   663.463, eps=0.000870857
Epoch 468: f=   663.460, eps=0.000914399
Epoch 469: f=   663.456, eps=0.000960119
Epoch 470: f=   663.455, eps=0.001008125
Epoch 471: f=   663.446, eps=0.001058532
Epoch 472: f=   663.438, eps=0.001111458
Epoch 473: f=   663.432, eps=0.001167031
Epoch 474: f=   663.428, eps=0.001225383
Epoch 475: f=   663.418, eps=0.001286652
Epoch 476: f=   663.417, eps=0.001350984
Epoch 477: f=   663.423, eps=0.000675492
Epoch 478: f=   663.418, eps=0.000709267
Epoch 479: f=   663.409, eps=0.000744730
Epoch 480: f=   663.403, eps=0.000781967
Epoch 481: f=   663.396, eps=0.000821065
Epoch 482: f=   663.393, eps=0.000862118
Epoch 483: f=   663.388, eps=0.000905224
Epoch 484: f=   663.384, eps=0.000950485
Epoch 485: f=   663.377, eps=0.000998010
```

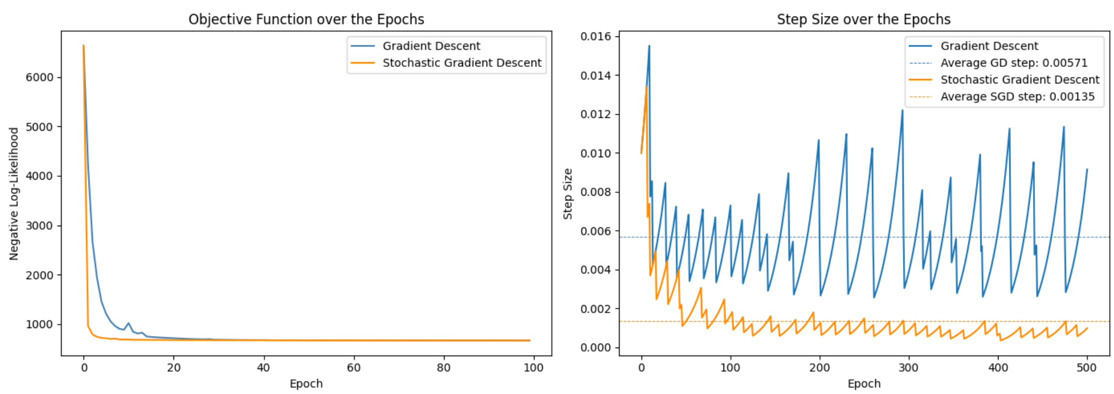
```
Epoch 486: f=   663.363, eps=0.001047910
Epoch 487: f=   663.362, eps=0.001100306
Epoch 488: f=   663.356, eps=0.001155321
Epoch 489: f=   663.358, eps=0.000577660
Epoch 490: f=   663.351, eps=0.000606543
Epoch 491: f=   663.343, eps=0.000636871
Epoch 492: f=   663.342, eps=0.000668714
Epoch 493: f=   663.339, eps=0.000702150
Epoch 494: f=   663.336, eps=0.000737257
Epoch 495: f=   663.333, eps=0.000774120
Epoch 496: f=   663.322, eps=0.000812826
Epoch 497: f=   663.317, eps=0.000853468
Epoch 498: f=   663.314, eps=0.000896141
Epoch 499: f=   663.311, eps=0.000940948
Result after 500 epochs: f=663.3023109198814
```

3.5 2e Compare GD and SGD

[135]:
```
# Color definitions for plotting
GD_MLE_COLOR = "steelblue"
SGD_MLE_COLOR = "darkorange"
```

[136]:
```
import matplotlib.pyplot as plt
import numpy as np

# Define the figure with two subplots side-by-side
plt.figure(figsize=(14, 5))

# 1st plot: Objective function over the epochs
plt.subplot(1, 2, 1)
plt.plot(vz_gd[0:100], label="Gradient Descent", color=GD_MLE_COLOR)
plt.plot(vz_sgd[0:100], label="Stochastic Gradient Descent",␣
 ↪color=SGD_MLE_COLOR)
plt.title("Objective Function over the Epochs")
plt.xlabel("Epoch")
plt.ylabel("Negative Log-Likelihood")
plt.legend()

# 2nd plot: Step size over the epochs
plt.subplot(1, 2, 2)
plt.plot(ez_gd, label="Gradient Descent")
plt.axhline(np.mean(ez_gd), color=GD_MLE_COLOR, linestyle='--', lw=0.7,␣
 ↪label=f'Average GD step: {np.mean(ez_gd):.5f}')
plt.plot(ez_sgd, label="Stochastic Gradient Descent", color=SGD_MLE_COLOR)
plt.axhline(np.mean(ez_sgd), color=SGD_MLE_COLOR, linestyle='--', lw=0.7,␣
 ↪label=f'Average SGD step: {np.mean(ez_sgd):.5f}')
plt.title("Step Size over the Epochs")
```

```python
plt.xlabel("Epoch")
plt.ylabel("Step Size")
plt.legend()

# Adjust layout for a tidy appearance
plt.tight_layout()
plt.show()

# Print summary statistics for GD and SGD log-likelihoods
print("Gradient Descent (GD) Summary:")
print(f"  Initial Log-likelihood: {vz_gd[0]:.4f}")
print(f"  Final Log-likelihood: {vz_gd[-1]:.4f}")
print(f"  Min Log-likelihood: {min(vz_gd):.4f}")

print("\nStochastic Gradient Descent (SGD) Summary:")
print(f"  Initial Log-likelihood: {vz_sgd[0]:.4f}")
print(f"  Final Log-likelihood: {vz_sgd[-1]:.4f}")
print(f"  Min Log-likelihood: {min(vz_sgd):.4f}")
```




```
Gradient Descent (GD) Summary:
  Initial Log-likelihood: 6636.2084
  Final Log-likelihood: 655.4135
  Min Log-likelihood: 655.4135

Stochastic Gradient Descent (SGD) Summary:
  Initial Log-likelihood: 6636.2084
  Final Log-likelihood: 663.3023
  Min Log-likelihood: 663.3023
```

4 3 Prediction

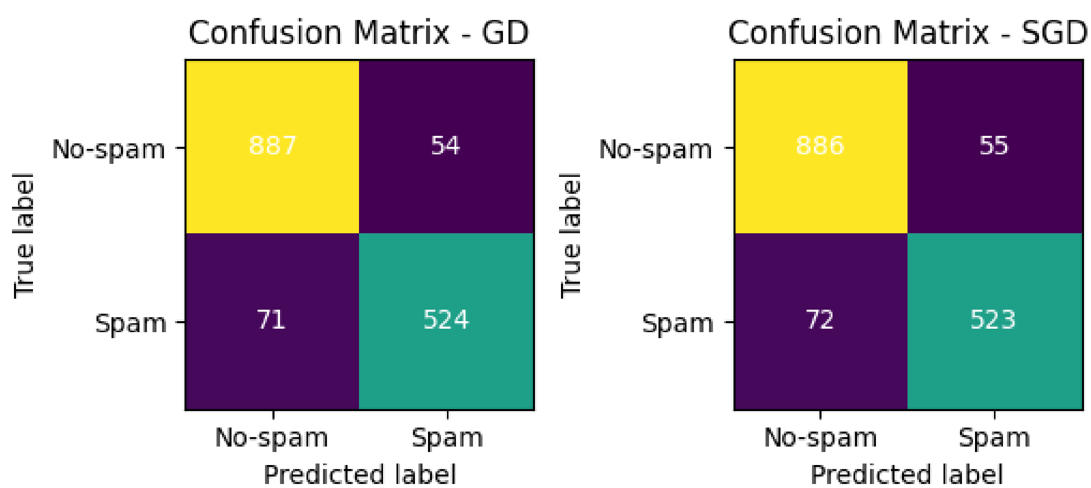
[137]:
```
def predict(Xtest, w):
    """Returns vector of predicted confidence values for logistic regression␣
↪with
weight vector w."""
    return sigma(Xtest @ w)


def classify(Xtest, w):
    """Returns 0/1 vector of predicted class labels for logistic regression with
weight vector w."""
    labels = (predict(Xtest, w) >= 0.5).astype(int)  # Convert boolean to int␣
↪(0 or 1)

    return labels
```

4.0.1 Confusuion matrices. GD vs SGD

[138]:
```
# Confusion matrix for the model trained with GD
yhat_gd = predict(Xtestz, wz_gd)
ypred_gd = classify(Xtestz, wz_gd)

confusion_matrix_gd = sklearn.metrics.confusion_matrix(ytest, ypred_gd)
print(confusion_matrix_gd)

# Confusion matrix for the model trained with GD
yhat_sgd = predict(Xtestz, wz_sgd)
ypred_sgd = classify(Xtestz, wz_sgd)

confusion_matrix_sgd = sklearn.metrics.confusion_matrix(ytest, ypred_sgd)
print(confusion_matrix_sgd)


# plot both confusion matrices one near another
fig, axes = plt.subplots(1, 2, figsize=(6, 3))
# confustion matrix for the model trained with gradient descent
axes[0].imshow(confusion_matrix_gd, interpolation='nearest', cmap=plt.cm.
↪viridis)
axes[0].set_title('Confusion Matrix - GD')
axes[0].set_ylabel('True label')
axes[0].set_xlabel('Predicted label')
ticks = np.arange(2)
axes[0].set_xticks(ticks)
axes[0].set_yticks(ticks)
axes[0].set_xticklabels(['No-spam', 'Spam'])
axes[0].set_yticklabels(['No-spam', 'Spam'])
```

```
for i in range(2):
    for j in range(2):
        axes[0].text(j, i, confusion_matrix_gd[i, j], ha='center', va='center',␣
 ↪color='white' if confusion_matrix_gd[i, j] > 50 else 'black')

# confustion matrix for the model trained with stohastic gradient descent
axes[1].imshow(confusion_matrix_sgd, interpolation='nearest', cmap=plt.cm.
 ↪viridis)
axes[1].set_title('Confusion Matrix - SGD')
axes[1].set_ylabel('True label')
axes[1].set_xlabel('Predicted label')
axes[1].set_xticks(ticks)
axes[1].set_yticks(ticks)
axes[1].set_xticklabels(['No-spam', 'Spam'])
axes[1].set_yticklabels(['No-spam', 'Spam'])
for i in range(2):
    for j in range(2):
        axes[1].text(j, i, confusion_matrix_sgd[i, j], ha='center',␣
 ↪va='center', color='white' if confusion_matrix_sgd[i, j] > 50 else 'black')


plt.tight_layout()
plt.show()
```

```
[[887  54]
 [ 71 524]]
[[886  55]
 [ 72 523]]
```



4.0.2 Classification Report. GD vs SGD

[139]:
```
import matplotlib.pyplot as plt
import pandas as pd
import sklearn.metrics as metrics

# Generate classification reports for both models as dictionaries
report_gd = metrics.classification_report(ytest, ypred_gd, output_dict=True)
report_sgd = metrics.classification_report(ytest, ypred_sgd, output_dict=True)

# Convert the classification reports to DataFrames and round to 3 decimal places
report_gd_df = pd.DataFrame(report_gd).transpose().round(3)
report_sgd_df = pd.DataFrame(report_sgd).transpose().round(3)

# Set up colors for better, worse, and equal metrics
def highlight_diff(val_gd, val_sgd):
    if val_gd > val_sgd:
        return "background-color: lightgreen"  # GD better
    elif val_gd < val_sgd:
        return "background-color: lightcoral"  # SGD better
    else:
        return ""  # Equal

# Plot both classification reports side by side with conditional formatting
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Visualization for the GD classification report with conditional formatting
axes[0].axis('off')  # Turn off the axis grid
table_gd = axes[0].table(
    cellText=report_gd_df.values,
    colLabels=report_gd_df.columns,
    rowLabels=report_gd_df.index,
    cellLoc='center',
    loc='center'
)
axes[0].set_title("Classification Report - GD")

# Visualization for the SGD classification report with conditional formatting
axes[1].axis('off')  # Turn off the axis grid
table_sgd = axes[1].table(
    cellText=report_sgd_df.values,
    colLabels=report_sgd_df.columns,
    rowLabels=report_sgd_df.index,
    cellLoc='center',
    loc='center'
)
axes[1].set_title("Classification Report - SGD")
```

```python
# Apply highlighting based on comparison
for i, row_label in enumerate(report_gd_df.index):
    for j, col_label in enumerate(report_gd_df.columns):
        # Get GD and SGD values for comparison
        val_gd = report_gd_df.at[row_label, col_label]
        val_sgd = report_sgd_df.at[row_label, col_label]

        # Highlight GD table
        table_gd[(i+1, j)].set_facecolor("lightgreen" if val_gd > val_sgd else␣
 ↪"lightcoral" if val_gd < val_sgd else "white")

        # Highlight SGD table
        table_sgd[(i+1, j)].set_facecolor("lightcoral" if val_gd > val_sgd else␣
 ↪"lightgreen" if val_gd < val_sgd else "white")

plt.tight_layout()
plt.show()
```

Classification Report - GD

	precision	recall	f1-score	support
0	0.926	0.943	0.934	941.0
1	0.907	0.881	0.893	595.0
accuracy	0.919	0.919	0.919	0.919
macro avg	0.916	0.912	0.914	1536.0
weighted avg	0.918	0.919	0.918	1536.0

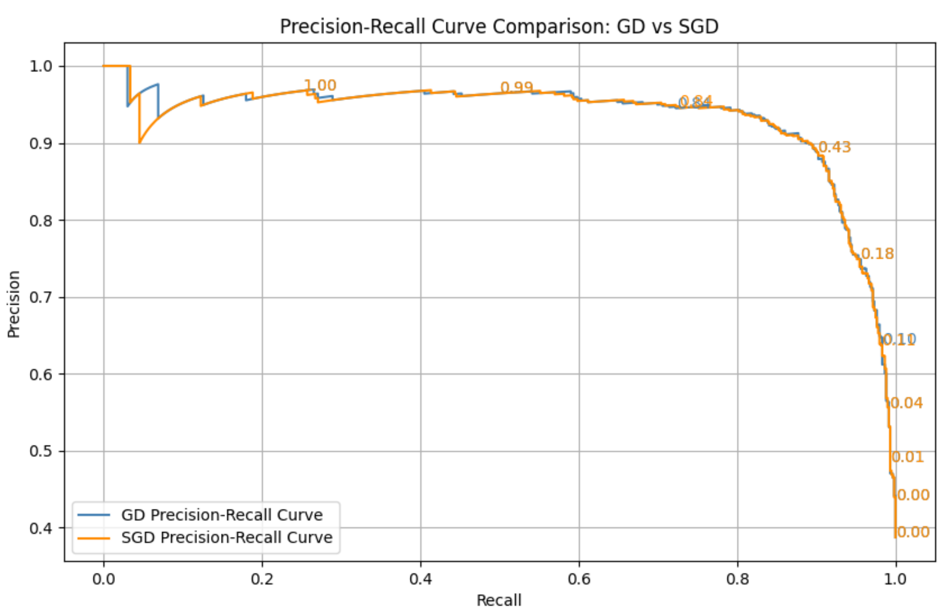
Classification Report - SGD

	precision	recall	f1-score	support
0	0.925	0.942	0.933	941.0
1	0.905	0.879	0.892	595.0
accuracy	0.917	0.917	0.917	0.917
macro avg	0.915	0.91	0.912	1536.0
weighted avg	0.917	0.917	0.917	1536.0

4.0.3 Precision-Recall Curve Comparison. GD vs SGD

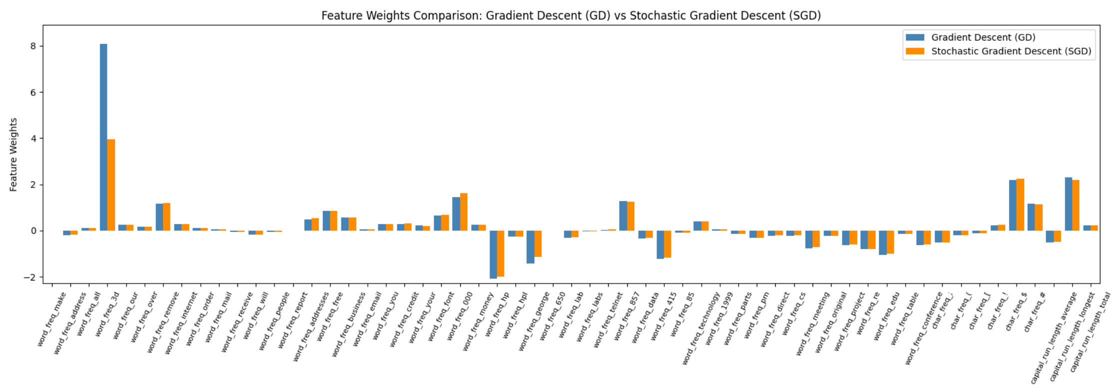
[140]:
```python
import numpy as np
import matplotlib.pyplot as plt
import sklearn.metrics as metrics

# Calculate precision, recall, and thresholds for both models
precision_gd, recall_gd, thresholds_gd = metrics.precision_recall_curve(ytest,␣
 ↪yhat_gd)
precision_sgd, recall_sgd, thresholds_sgd = metrics.
 ↪precision_recall_curve(ytest, yhat_sgd)
```

```python
# Plot precision-recall curves for both models
plt.figure(figsize=(10, 6))

# Plot for Gradient Descent (GD) with consistent color
plt.plot(recall_gd, precision_gd, label="GD Precision-Recall Curve",␣
 ↪color=GD_MLE_COLOR)
for x in np.linspace(0, 1, 10, endpoint=False):
    index_gd = int(x * (precision_gd.size - 1))
    plt.text(recall_gd[index_gd], precision_gd[index_gd], "{:3.2f}".
 ↪format(thresholds_gd[index_gd]), color=GD_MLE_COLOR)

# Plot for Stochastic Gradient Descent (SGD) with consistent color
plt.plot(recall_sgd, precision_sgd, label="SGD Precision-Recall Curve",␣
 ↪color=SGD_MLE_COLOR)
for x in np.linspace(0, 1, 10, endpoint=False):
    index_sgd = int(x * (precision_sgd.size - 1))
    plt.text(recall_sgd[index_sgd], precision_sgd[index_sgd], "{:3.2f}".
 ↪format(thresholds_sgd[index_sgd]), color=SGD_MLE_COLOR)

# Labels and title
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve Comparison: GD vs SGD")
plt.legend()
plt.grid(True)
plt.show()
```



4.0.4 Feature Importance. GD vs SGD

[141]:
```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(figsize=(len(features) * 0.3, 6))
x_positions = np.arange(len(features))

# Plot weights with consistent color and labels
ax.bar(x_positions - 0.2, wz_gd, width=0.4, color=GD_MLE_COLOR, label="Gradient␣
↪Descent (GD)")
ax.bar(x_positions + 0.2, wz_sgd, width=0.4, color=SGD_MLE_COLOR,␣
↪label="Stochastic Gradient Descent (SGD)")

# Set feature names on the x-axis as ticks
ax.set_xticks(x_positions)
ax.set_xticklabels(features, rotation=65, fontsize=8)
ax.set_xlim(-0.5, len(features) - 0.5)

# Add labels and title
ax.set_ylabel("Feature Weights")
plt.title("Feature Weights Comparison: Gradient Descent (GD) vs Stochastic␣
↪Gradient Descent (SGD)")

# Set y-axis range to include both negative and positive weight values
min_weight = min(np.min(wz_gd), np.min(wz_sgd))
max_weight = max(np.max(wz_gd), np.max(wz_sgd))
ax.set_ylim([min_weight * 1.1, max_weight * 1.1])

# Add a legend and tighten layout
ax.legend()
plt.tight_layout()

# Display the plot
plt.show()
```



5 4 Maximum Aposteriori Estimation

5.1 4a Gradient Descent

5.1.1 Log-density of a posterior

Formula is taken from: 05 lecture. Part 3. Slide 9/12

[142]:
```
def l_l2(y, X, w, lambda_):
    """Log-density of posterior of logistic regression with weights w and L2
regularization parameter lambda_"""
    #computing the log-likelihood
    log_likelihood = l(y, X, w)

    # introducing L2 regularization
    l2_penalty = lambda_/2 * (np.linalg.norm(w) ** 2)

    # applying it to log-likelihood
    return log_likelihood - l2_penalty
```

[143]:
```
# this should give:
# [-47066.641667825766, -47312.623810682911]
[l_l2(y, Xz, np.linspace(-5, 5, D), 0), l_l2(y, Xz, np.linspace(-5, 5, D), 1)]
```

[143]: [-47066.641667825774, -47312.62381068292]

5.1.2 Gradient of log-density of posterior

Formula is derived by taking the partial derivative w.r.t. w

[144]:
```
def dl_l2(y, X, w, lambda_):
    """Gradient of log-density of posterior of logistic regression with weights␣
 ↪w
and L2 regularization parameter lambda_."""
```

```python
    # computing the gradient of log-likelihood
    gradient_likelihood = dl(y, X, w)

    # computing the gradient of the L2 regularization term
    gradient_l2 = lambda_ * w

    # applying it to log-likelihood gradient
    return gradient_likelihood - gradient_l2
```

[145]:
```python
# this should give:
# [array([  551.33985842,   143.84116318,   841.83373606,   156.87237578,
#           802.61217579,   795.96202907,   920.69045803,   621.96516752,
#           659.18724769,   470.81259805,   771.32406968,   352.40325626,
#           455.66972482,   234.36600888,   562.45454038,   864.83981264,
#           787.19723703,   649.48042176,   902.6478154 ,   544.00539886,
#          1174.78638035,   120.3598967 ,   839.61141672,   633.30453444,
#          -706.66815087,  -630.2039816 ,  -569.3451386 ,  -527.50996698,
#          -359.53701083,  -476.64334832,  -411.60620464,  -375.11950586,
#          -345.37195689,  -376.22044258,  -407.31761977,  -456.23251936,
#          -596.86960184,  -107.97072355,  -394.82170044,  -229.18125598,
#          -288.46356547,  -362.13402385,  -450.87896465,  -277.03932676,
#          -414.99293368,  -452.28771693,  -167.54649092,  -270.9043748 ,
#          -252.20140951,  -357.72497343,  -259.12468742,   418.35938483,
#           604.54173228,    43.10390907,   152.24258478,   378.16731033,
#           416.12032881]),
#  array([  556.33985842,   148.66259175,   846.4765932 ,   161.33666149,
#           806.89789007,   800.06917193,   924.61902946,   625.71516752,
#           662.75867626,   474.20545519,   774.5383554 ,   355.43897054,
#           458.52686767,   237.04458031,   564.95454038,   867.16124121,
#           789.34009417,   651.44470748,   904.43352968,   545.61254171,
#          1176.21495178,   121.6098967 ,   840.68284529,   634.19739158,
#          -705.95386516,  -629.66826731,  -568.98799574,  -527.33139555,
#          -359.53701083,  -476.82191975,  -411.9633475 ,  -375.65522015,
#          -346.08624261,  -377.11329972,  -408.38904835,  -457.48251936,
#          -598.29817327,  -109.57786641,  -396.60741472,  -231.14554169,
#          -290.60642261,  -364.45545242,  -453.37896465,  -279.71789819,
#          -417.85007654,  -455.32343122,  -170.76077664,  -274.29723194,
#          -255.77283808,  -361.47497343,  -263.05325885,   414.25224198,
#           600.25601799,    38.63962335,   147.59972763,   373.34588176,
#           411.12032881])]
[dl_l2(y, Xz, np.linspace(-5, 5, D), 0), dl_l2(y, Xz, np.linspace(-5, 5, D), 1)]
```

[145]:
```
[array([ 551.33985842,  143.84116318,  841.83373606,  156.87237578,
         802.61217579,  795.96202907,  920.69045803,  621.96516752,
         659.18724769,  470.81259805,  771.32406968,  352.40325626,
         455.66972482,  234.36600888,  562.45454038,  864.83981264,
```

```
        787.19723703,  649.48042176,  902.6478154 ,  544.00539886,
       1174.78638035,  120.3598967 ,  839.61141672,  633.30453444,
       -706.66815087, -630.2039816 , -569.3451386 , -527.50996698,
       -359.53701083, -476.64334832, -411.60620464, -375.11950586,
       -345.37195689, -376.22044258, -407.31761977, -456.23251936,
       -596.86960184, -107.97072355, -394.82170044, -229.18125598,
       -288.46356547, -362.13402385, -450.87896465, -277.03932676,
       -414.99293368, -452.28771693, -167.54649092, -270.9043748 ,
       -252.20140951, -357.72497343, -259.12468742,  418.35938483,
        604.54173228,   43.10390907,  152.24258478,  378.16731033,
        416.12032881]),
 array([ 556.33985842,  148.66259175,  846.4765932 ,  161.33666149,
        806.89789007,  800.06917193,  924.61902946,  625.71516752,
        662.75867626,  474.20545519,  774.5383554 ,  355.43897054,
        458.52686767,  237.04458031,  564.95454038,  867.16124121,
        789.34009417,  651.44470748,  904.43352968,  545.61254171,
       1176.21495178,  121.6098967 ,  840.68284529,  634.19739158,
       -705.95386516, -629.66826731, -568.98799574, -527.33139555,
       -359.53701083, -476.82191975, -411.9633475 , -375.65522015,
       -346.08624261, -377.11329972, -408.38904835, -457.48251936,
       -598.29817327, -109.57786641, -396.60741472, -231.14554169,
       -290.60642261, -364.45545242, -453.37896465, -279.71789819,
       -417.85007654, -455.32343122, -170.76077664, -274.29723194,
       -255.77283808, -361.47497343, -263.05325885,  414.25224198,
        600.25601799,   38.63962335,  147.59972763,  373.34588176,
        411.12032881])]
```

5.1.3 Gradient descent for MAP estimation with L2 regularization

[146]:
```python
# now define the (f,update) tuple for optimize for logistic regression, L2
# regularization, and gradient descent
def gd_l2(y, X, lambda_):
    def objective(w):
        return -l_l2(y, X, w, lambda_)

    def update(w, eps):
        grad = dl_l2(y, X, w, lambda_)
        return w + eps * grad  # Gradient ascent step (maximize posterior)

    return (objective, update)
```

[147]:
```python
# let's run!
lambda_ = 100
wz_gd_l2, vz_gd_l2, ez_gd_l2 = optimize(gd_l2(y, Xz, lambda_), w0, nepochs=500)
```

```
Epoch   0: f=  9992.358, eps=0.010000000
Epoch   1: f= 23977.384, eps=0.005000000
```

```
Epoch   2: f=  5534.851, eps=0.005250000
Epoch   3: f=  1427.453, eps=0.005512500
Epoch   4: f=  1131.716, eps=0.005788125
Epoch   5: f=  1540.933, eps=0.002894063
Epoch   6: f=  1323.168, eps=0.003038766
Epoch   7: f=  1049.068, eps=0.003190704
Epoch   8: f=  1067.960, eps=0.001595352
Epoch   9: f=   989.861, eps=0.001675120
Epoch  10: f=   988.742, eps=0.001758876
Epoch  11: f=   988.585, eps=0.001846819
Epoch  12: f=   988.539, eps=0.001939160
Epoch  13: f=   988.522, eps=0.002036118
Epoch  14: f=   988.516, eps=0.002137924
Epoch  15: f=   988.513, eps=0.002244820
Epoch  16: f=   988.512, eps=0.002357061
Epoch  17: f=   988.512, eps=0.002474914
Epoch  18: f=   988.512, eps=0.002598660
Epoch  19: f=   988.512, eps=0.002728593
Epoch  20: f=   988.512, eps=0.002865023
Epoch  21: f=   988.512, eps=0.003008274
Epoch  22: f=   988.512, eps=0.003158688
Epoch  23: f=   988.512, eps=0.003316622
Epoch  24: f=   988.512, eps=0.001658311
Epoch  25: f=   988.512, eps=0.001741227
Epoch  26: f=   988.512, eps=0.001828288
Epoch  27: f=   988.512, eps=0.001919702
Epoch  28: f=   988.512, eps=0.002015687
Epoch  29: f=   988.512, eps=0.002116472
Epoch  30: f=   988.512, eps=0.002222295
Epoch  31: f=   988.512, eps=0.002333410
Epoch  32: f=   988.512, eps=0.002450081
Epoch  33: f=   988.512, eps=0.002572585
Epoch  34: f=   988.512, eps=0.002701214
Epoch  35: f=   988.512, eps=0.002836275
Epoch  36: f=   988.512, eps=0.002978088
Epoch  37: f=   988.512, eps=0.003126993
Epoch  38: f=   988.512, eps=0.003283342
Epoch  39: f=   988.512, eps=0.003447510
Epoch  40: f=   988.512, eps=0.003619885
Epoch  41: f=   988.512, eps=0.001809943
Epoch  42: f=   988.512, eps=0.001900440
Epoch  43: f=   988.512, eps=0.001995462
Epoch  44: f=   988.512, eps=0.002095235
Epoch  45: f=   988.512, eps=0.002199996
Epoch  46: f=   988.512, eps=0.002309996
Epoch  47: f=   988.512, eps=0.002425496
Epoch  48: f=   988.512, eps=0.001212748
Epoch  49: f=   988.512, eps=0.001273385
```

```
Epoch  50: f=   988.512, eps=0.001337055
Epoch  51: f=   988.512, eps=0.001403907
Epoch  52: f=   988.512, eps=0.001474103
Epoch  53: f=   988.512, eps=0.000737051
Epoch  54: f=   988.512, eps=0.000773904
Epoch  55: f=   988.512, eps=0.000812599
Epoch  56: f=   988.512, eps=0.000406300
Epoch  57: f=   988.512, eps=0.000426615
Epoch  58: f=   988.512, eps=0.000447945
Epoch  59: f=   988.512, eps=0.000223973
Epoch  60: f=   988.512, eps=0.000235171
Epoch  61: f=   988.512, eps=0.000117586
Epoch  62: f=   988.512, eps=0.000123465
Epoch  63: f=   988.512, eps=0.000061732
Epoch  64: f=   988.512, eps=0.000064819
Epoch  65: f=   988.512, eps=0.000068060
Epoch  66: f=   988.512, eps=0.000071463
Epoch  67: f=   988.512, eps=0.000075036
Epoch  68: f=   988.512, eps=0.000078788
Epoch  69: f=   988.512, eps=0.000039394
Epoch  70: f=   988.512, eps=0.000041364
Epoch  71: f=   988.512, eps=0.000043432
Epoch  72: f=   988.512, eps=0.000045603
Epoch  73: f=   988.512, eps=0.000022802
Epoch  74: f=   988.512, eps=0.000023942
Epoch  75: f=   988.512, eps=0.000025139
Epoch  76: f=   988.512, eps=0.000026396
Epoch  77: f=   988.512, eps=0.000013198
Epoch  78: f=   988.512, eps=0.000013858
Epoch  79: f=   988.512, eps=0.000014551
Epoch  80: f=   988.512, eps=0.000015278
Epoch  81: f=   988.512, eps=0.000007639
Epoch  82: f=   988.512, eps=0.000008021
Epoch  83: f=   988.512, eps=0.000008422
Epoch  84: f=   988.512, eps=0.000008843
Epoch  85: f=   988.512, eps=0.000004422
Epoch  86: f=   988.512, eps=0.000004643
Epoch  87: f=   988.512, eps=0.000004875
Epoch  88: f=   988.512, eps=0.000005119
Epoch  89: f=   988.512, eps=0.000002559
Epoch  90: f=   988.512, eps=0.000002687
Epoch  91: f=   988.512, eps=0.000002822
Epoch  92: f=   988.512, eps=0.000001411
Epoch  93: f=   988.512, eps=0.000001481
Epoch  94: f=   988.512, eps=0.000001555
Epoch  95: f=   988.512, eps=0.000000778
Epoch  96: f=   988.512, eps=0.000000817
Epoch  97: f=   988.512, eps=0.000000857
```

```
Epoch  98: f=   988.512, eps=0.000000900
Epoch  99: f=   988.512, eps=0.000000945
Epoch 100: f=   988.512, eps=0.000000993
Epoch 101: f=   988.512, eps=0.000001042
Epoch 102: f=   988.512, eps=0.000000521
Epoch 103: f=   988.512, eps=0.000000547
Epoch 104: f=   988.512, eps=0.000000575
Epoch 105: f=   988.512, eps=0.000000603
Epoch 106: f=   988.512, eps=0.000000302
Epoch 107: f=   988.512, eps=0.000000317
Epoch 108: f=   988.512, eps=0.000000333
Epoch 109: f=   988.512, eps=0.000000349
Epoch 110: f=   988.512, eps=0.000000367
Epoch 111: f=   988.512, eps=0.000000385
Epoch 112: f=   988.512, eps=0.000000192
Epoch 113: f=   988.512, eps=0.000000202
Epoch 114: f=   988.512, eps=0.000000101
Epoch 115: f=   988.512, eps=0.000000106
Epoch 116: f=   988.512, eps=0.000000053
Epoch 117: f=   988.512, eps=0.000000056
Epoch 118: f=   988.512, eps=0.000000058
Epoch 119: f=   988.512, eps=0.000000029
Epoch 120: f=   988.512, eps=0.000000031
Epoch 121: f=   988.512, eps=0.000000032
Epoch 122: f=   988.512, eps=0.000000034
Epoch 123: f=   988.512, eps=0.000000036
Epoch 124: f=   988.512, eps=0.000000018
Epoch 125: f=   988.512, eps=0.000000019
Epoch 126: f=   988.512, eps=0.000000020
Epoch 127: f=   988.512, eps=0.000000010
Epoch 128: f=   988.512, eps=0.000000010
Epoch 129: f=   988.512, eps=0.000000005
Epoch 130: f=   988.512, eps=0.000000005
Epoch 131: f=   988.512, eps=0.000000006
Epoch 132: f=   988.512, eps=0.000000006
Epoch 133: f=   988.512, eps=0.000000006
Epoch 134: f=   988.512, eps=0.000000007
Epoch 135: f=   988.512, eps=0.000000003
Epoch 136: f=   988.512, eps=0.000000003
Epoch 137: f=   988.512, eps=0.000000004
Epoch 138: f=   988.512, eps=0.000000004
Epoch 139: f=   988.512, eps=0.000000002
Epoch 140: f=   988.512, eps=0.000000002
Epoch 141: f=   988.512, eps=0.000000002
Epoch 142: f=   988.512, eps=0.000000002
Epoch 143: f=   988.512, eps=0.000000002
Epoch 144: f=   988.512, eps=0.000000002
Epoch 145: f=   988.512, eps=0.000000003
```

```
Epoch 146: f=   988.512, eps=0.000000003
Epoch 147: f=   988.512, eps=0.000000003
Epoch 148: f=   988.512, eps=0.000000003
Epoch 149: f=   988.512, eps=0.000000003
Epoch 150: f=   988.512, eps=0.000000003
Epoch 151: f=   988.512, eps=0.000000003
Epoch 152: f=   988.512, eps=0.000000002
Epoch 153: f=   988.512, eps=0.000000002
Epoch 154: f=   988.512, eps=0.000000001
Epoch 155: f=   988.512, eps=0.000000001
Epoch 156: f=   988.512, eps=0.000000001
Epoch 157: f=   988.512, eps=0.000000001
Epoch 158: f=   988.512, eps=0.000000001
Epoch 159: f=   988.512, eps=0.000000001
Epoch 160: f=   988.512, eps=0.000000001
Epoch 161: f=   988.512, eps=0.000000001
Epoch 162: f=   988.512, eps=0.000000001
Epoch 163: f=   988.512, eps=0.000000001
Epoch 164: f=   988.512, eps=0.000000001
Epoch 165: f=   988.512, eps=0.000000002
Epoch 166: f=   988.512, eps=0.000000002
Epoch 167: f=   988.512, eps=0.000000001
Epoch 168: f=   988.512, eps=0.000000001
Epoch 169: f=   988.512, eps=0.000000001
Epoch 170: f=   988.512, eps=0.000000000
Epoch 171: f=   988.512, eps=0.000000000
Epoch 172: f=   988.512, eps=0.000000000
Epoch 173: f=   988.512, eps=0.000000001
Epoch 174: f=   988.512, eps=0.000000001
Epoch 175: f=   988.512, eps=0.000000001
Epoch 176: f=   988.512, eps=0.000000000
Epoch 177: f=   988.512, eps=0.000000000
Epoch 178: f=   988.512, eps=0.000000000
Epoch 179: f=   988.512, eps=0.000000000
Epoch 180: f=   988.512, eps=0.000000000
Epoch 181: f=   988.512, eps=0.000000000
Epoch 182: f=   988.512, eps=0.000000000
Epoch 183: f=   988.512, eps=0.000000000
Epoch 184: f=   988.512, eps=0.000000000
Epoch 185: f=   988.512, eps=0.000000000
Epoch 186: f=   988.512, eps=0.000000000
Epoch 187: f=   988.512, eps=0.000000000
Epoch 188: f=   988.512, eps=0.000000000
Epoch 189: f=   988.512, eps=0.000000000
Epoch 190: f=   988.512, eps=0.000000000
Epoch 191: f=   988.512, eps=0.000000000
Epoch 192: f=   988.512, eps=0.000000000
Epoch 193: f=   988.512, eps=0.000000000
```

```
Epoch 194: f=   988.512, eps=0.000000000
Epoch 195: f=   988.512, eps=0.000000000
Epoch 196: f=   988.512, eps=0.000000000
Epoch 197: f=   988.512, eps=0.000000000
Epoch 198: f=   988.512, eps=0.000000000
Epoch 199: f=   988.512, eps=0.000000000
Epoch 200: f=   988.512, eps=0.000000000
Epoch 201: f=   988.512, eps=0.000000000
Epoch 202: f=   988.512, eps=0.000000000
Epoch 203: f=   988.512, eps=0.000000001
Epoch 204: f=   988.512, eps=0.000000001
Epoch 205: f=   988.512, eps=0.000000001
Epoch 206: f=   988.512, eps=0.000000001
Epoch 207: f=   988.512, eps=0.000000001
Epoch 208: f=   988.512, eps=0.000000001
Epoch 209: f=   988.512, eps=0.000000001
Epoch 210: f=   988.512, eps=0.000000000
Epoch 211: f=   988.512, eps=0.000000000
Epoch 212: f=   988.512, eps=0.000000000
Epoch 213: f=   988.512, eps=0.000000000
Epoch 214: f=   988.512, eps=0.000000000
Epoch 215: f=   988.512, eps=0.000000000
Epoch 216: f=   988.512, eps=0.000000000
Epoch 217: f=   988.512, eps=0.000000000
Epoch 218: f=   988.512, eps=0.000000000
Epoch 219: f=   988.512, eps=0.000000000
Epoch 220: f=   988.512, eps=0.000000000
Epoch 221: f=   988.512, eps=0.000000000
Epoch 222: f=   988.512, eps=0.000000000
Epoch 223: f=   988.512, eps=0.000000000
Epoch 224: f=   988.512, eps=0.000000000
Epoch 225: f=   988.512, eps=0.000000000
Epoch 226: f=   988.512, eps=0.000000000
Epoch 227: f=   988.512, eps=0.000000000
Epoch 228: f=   988.512, eps=0.000000000
Epoch 229: f=   988.512, eps=0.000000000
Epoch 230: f=   988.512, eps=0.000000000
Epoch 231: f=   988.512, eps=0.000000000
Epoch 232: f=   988.512, eps=0.000000000
Epoch 233: f=   988.512, eps=0.000000000
Epoch 234: f=   988.512, eps=0.000000000
Epoch 235: f=   988.512, eps=0.000000000
Epoch 236: f=   988.512, eps=0.000000000
Epoch 237: f=   988.512, eps=0.000000000
Epoch 238: f=   988.512, eps=0.000000000
Epoch 239: f=   988.512, eps=0.000000000
Epoch 240: f=   988.512, eps=0.000000000
Epoch 241: f=   988.512, eps=0.000000000
```

```
Epoch 242: f=   988.512, eps=0.000000000
Epoch 243: f=   988.512, eps=0.000000000
Epoch 244: f=   988.512, eps=0.000000000
Epoch 245: f=   988.512, eps=0.000000000
Epoch 246: f=   988.512, eps=0.000000000
Epoch 247: f=   988.512, eps=0.000000000
Epoch 248: f=   988.512, eps=0.000000000
Epoch 249: f=   988.512, eps=0.000000000
Epoch 250: f=   988.512, eps=0.000000000
Epoch 251: f=   988.512, eps=0.000000000
Epoch 252: f=   988.512, eps=0.000000000
Epoch 253: f=   988.512, eps=0.000000000
Epoch 254: f=   988.512, eps=0.000000000
Epoch 255: f=   988.512, eps=0.000000000
Epoch 256: f=   988.512, eps=0.000000000
Epoch 257: f=   988.512, eps=0.000000000
Epoch 258: f=   988.512, eps=0.000000000
Epoch 259: f=   988.512, eps=0.000000000
Epoch 260: f=   988.512, eps=0.000000000
Epoch 261: f=   988.512, eps=0.000000000
Epoch 262: f=   988.512, eps=0.000000000
Epoch 263: f=   988.512, eps=0.000000000
Epoch 264: f=   988.512, eps=0.000000000
Epoch 265: f=   988.512, eps=0.000000000
Epoch 266: f=   988.512, eps=0.000000000
Epoch 267: f=   988.512, eps=0.000000000
Epoch 268: f=   988.512, eps=0.000000000
Epoch 269: f=   988.512, eps=0.000000000
Epoch 270: f=   988.512, eps=0.000000000
Epoch 271: f=   988.512, eps=0.000000000
Epoch 272: f=   988.512, eps=0.000000000
Epoch 273: f=   988.512, eps=0.000000000
Epoch 274: f=   988.512, eps=0.000000000
Epoch 275: f=   988.512, eps=0.000000000
Epoch 276: f=   988.512, eps=0.000000000
Epoch 277: f=   988.512, eps=0.000000000
Epoch 278: f=   988.512, eps=0.000000000
Epoch 279: f=   988.512, eps=0.000000000
Epoch 280: f=   988.512, eps=0.000000000
Epoch 281: f=   988.512, eps=0.000000000
Epoch 282: f=   988.512, eps=0.000000000
Epoch 283: f=   988.512, eps=0.000000000
Epoch 284: f=   988.512, eps=0.000000000
Epoch 285: f=   988.512, eps=0.000000000
Epoch 286: f=   988.512, eps=0.000000000
Epoch 287: f=   988.512, eps=0.000000000
Epoch 288: f=   988.512, eps=0.000000000
Epoch 289: f=   988.512, eps=0.000000000
```

```
Epoch 290: f=   988.512, eps=0.000000000
Epoch 291: f=   988.512, eps=0.000000000
Epoch 292: f=   988.512, eps=0.000000000
Epoch 293: f=   988.512, eps=0.000000000
Epoch 294: f=   988.512, eps=0.000000000
Epoch 295: f=   988.512, eps=0.000000000
Epoch 296: f=   988.512, eps=0.000000000
Epoch 297: f=   988.512, eps=0.000000000
Epoch 298: f=   988.512, eps=0.000000000
Epoch 299: f=   988.512, eps=0.000000000
Epoch 300: f=   988.512, eps=0.000000000
Epoch 301: f=   988.512, eps=0.000000000
Epoch 302: f=   988.512, eps=0.000000000
Epoch 303: f=   988.512, eps=0.000000000
Epoch 304: f=   988.512, eps=0.000000000
Epoch 305: f=   988.512, eps=0.000000000
Epoch 306: f=   988.512, eps=0.000000000
Epoch 307: f=   988.512, eps=0.000000000
Epoch 308: f=   988.512, eps=0.000000000
Epoch 309: f=   988.512, eps=0.000000000
Epoch 310: f=   988.512, eps=0.000000001
Epoch 311: f=   988.512, eps=0.000000001
Epoch 312: f=   988.512, eps=0.000000001
Epoch 313: f=   988.512, eps=0.000000001
Epoch 314: f=   988.512, eps=0.000000001
Epoch 315: f=   988.512, eps=0.000000001
Epoch 316: f=   988.512, eps=0.000000001
Epoch 317: f=   988.512, eps=0.000000001
Epoch 318: f=   988.512, eps=0.000000001
Epoch 319: f=   988.512, eps=0.000000001
Epoch 320: f=   988.512, eps=0.000000001
Epoch 321: f=   988.512, eps=0.000000000
Epoch 322: f=   988.512, eps=0.000000000
Epoch 323: f=   988.512, eps=0.000000000
Epoch 324: f=   988.512, eps=0.000000000
Epoch 325: f=   988.512, eps=0.000000001
Epoch 326: f=   988.512, eps=0.000000000
Epoch 327: f=   988.512, eps=0.000000000
Epoch 328: f=   988.512, eps=0.000000000
Epoch 329: f=   988.512, eps=0.000000000
Epoch 330: f=   988.512, eps=0.000000000
Epoch 331: f=   988.512, eps=0.000000000
Epoch 332: f=   988.512, eps=0.000000000
Epoch 333: f=   988.512, eps=0.000000000
Epoch 334: f=   988.512, eps=0.000000000
Epoch 335: f=   988.512, eps=0.000000000
Epoch 336: f=   988.512, eps=0.000000000
Epoch 337: f=   988.512, eps=0.000000000
```

```
Epoch 338: f=   988.512, eps=0.000000000
Epoch 339: f=   988.512, eps=0.000000000
Epoch 340: f=   988.512, eps=0.000000000
Epoch 341: f=   988.512, eps=0.000000000
Epoch 342: f=   988.512, eps=0.000000000
Epoch 343: f=   988.512, eps=0.000000000
Epoch 344: f=   988.512, eps=0.000000000
Epoch 345: f=   988.512, eps=0.000000000
Epoch 346: f=   988.512, eps=0.000000000
Epoch 347: f=   988.512, eps=0.000000000
Epoch 348: f=   988.512, eps=0.000000000
Epoch 349: f=   988.512, eps=0.000000000
Epoch 350: f=   988.512, eps=0.000000000
Epoch 351: f=   988.512, eps=0.000000000
Epoch 352: f=   988.512, eps=0.000000000
Epoch 353: f=   988.512, eps=0.000000000
Epoch 354: f=   988.512, eps=0.000000000
Epoch 355: f=   988.512, eps=0.000000000
Epoch 356: f=   988.512, eps=0.000000000
Epoch 357: f=   988.512, eps=0.000000000
Epoch 358: f=   988.512, eps=0.000000000
Epoch 359: f=   988.512, eps=0.000000000
Epoch 360: f=   988.512, eps=0.000000000
Epoch 361: f=   988.512, eps=0.000000000
Epoch 362: f=   988.512, eps=0.000000000
Epoch 363: f=   988.512, eps=0.000000000
Epoch 364: f=   988.512, eps=0.000000000
Epoch 365: f=   988.512, eps=0.000000000
Epoch 366: f=   988.512, eps=0.000000000
Epoch 367: f=   988.512, eps=0.000000000
Epoch 368: f=   988.512, eps=0.000000000
Epoch 369: f=   988.512, eps=0.000000000
Epoch 370: f=   988.512, eps=0.000000000
Epoch 371: f=   988.512, eps=0.000000000
Epoch 372: f=   988.512, eps=0.000000000
Epoch 373: f=   988.512, eps=0.000000000
Epoch 374: f=   988.512, eps=0.000000000
Epoch 375: f=   988.512, eps=0.000000000
Epoch 376: f=   988.512, eps=0.000000000
Epoch 377: f=   988.512, eps=0.000000000
Epoch 378: f=   988.512, eps=0.000000000
Epoch 379: f=   988.512, eps=0.000000000
Epoch 380: f=   988.512, eps=0.000000000
Epoch 381: f=   988.512, eps=0.000000000
Epoch 382: f=   988.512, eps=0.000000000
Epoch 383: f=   988.512, eps=0.000000000
Epoch 384: f=   988.512, eps=0.000000000
Epoch 385: f=   988.512, eps=0.000000000
```

```
Epoch 386: f=   988.512, eps=0.000000000
Epoch 387: f=   988.512, eps=0.000000000
Epoch 388: f=   988.512, eps=0.000000000
Epoch 389: f=   988.512, eps=0.000000000
Epoch 390: f=   988.512, eps=0.000000000
Epoch 391: f=   988.512, eps=0.000000000
Epoch 392: f=   988.512, eps=0.000000000
Epoch 393: f=   988.512, eps=0.000000000
Epoch 394: f=   988.512, eps=0.000000000
Epoch 395: f=   988.512, eps=0.000000000
Epoch 396: f=   988.512, eps=0.000000000
Epoch 397: f=   988.512, eps=0.000000000
Epoch 398: f=   988.512, eps=0.000000000
Epoch 399: f=   988.512, eps=0.000000000
Epoch 400: f=   988.512, eps=0.000000000
Epoch 401: f=   988.512, eps=0.000000000
Epoch 402: f=   988.512, eps=0.000000000
Epoch 403: f=   988.512, eps=0.000000000
Epoch 404: f=   988.512, eps=0.000000000
Epoch 405: f=   988.512, eps=0.000000000
Epoch 406: f=   988.512, eps=0.000000000
Epoch 407: f=   988.512, eps=0.000000000
Epoch 408: f=   988.512, eps=0.000000000
Epoch 409: f=   988.512, eps=0.000000000
Epoch 410: f=   988.512, eps=0.000000000
Epoch 411: f=   988.512, eps=0.000000000
Epoch 412: f=   988.512, eps=0.000000000
Epoch 413: f=   988.512, eps=0.000000000
Epoch 414: f=   988.512, eps=0.000000000
Epoch 415: f=   988.512, eps=0.000000000
Epoch 416: f=   988.512, eps=0.000000000
Epoch 417: f=   988.512, eps=0.000000000
Epoch 418: f=   988.512, eps=0.000000000
Epoch 419: f=   988.512, eps=0.000000000
Epoch 420: f=   988.512, eps=0.000000000
Epoch 421: f=   988.512, eps=0.000000000
Epoch 422: f=   988.512, eps=0.000000000
Epoch 423: f=   988.512, eps=0.000000000
Epoch 424: f=   988.512, eps=0.000000000
Epoch 425: f=   988.512, eps=0.000000000
Epoch 426: f=   988.512, eps=0.000000000
Epoch 427: f=   988.512, eps=0.000000000
Epoch 428: f=   988.512, eps=0.000000000
Epoch 429: f=   988.512, eps=0.000000000
Epoch 430: f=   988.512, eps=0.000000000
Epoch 431: f=   988.512, eps=0.000000000
Epoch 432: f=   988.512, eps=0.000000000
Epoch 433: f=   988.512, eps=0.000000000
```

```
Epoch 434: f=   988.512, eps=0.000000000
Epoch 435: f=   988.512, eps=0.000000000
Epoch 436: f=   988.512, eps=0.000000000
Epoch 437: f=   988.512, eps=0.000000000
Epoch 438: f=   988.512, eps=0.000000000
Epoch 439: f=   988.512, eps=0.000000000
Epoch 440: f=   988.512, eps=0.000000000
Epoch 441: f=   988.512, eps=0.000000000
Epoch 442: f=   988.512, eps=0.000000000
Epoch 443: f=   988.512, eps=0.000000000
Epoch 444: f=   988.512, eps=0.000000000
Epoch 445: f=   988.512, eps=0.000000000
Epoch 446: f=   988.512, eps=0.000000000
Epoch 447: f=   988.512, eps=0.000000000
Epoch 448: f=   988.512, eps=0.000000000
Epoch 449: f=   988.512, eps=0.000000000
Epoch 450: f=   988.512, eps=0.000000000
Epoch 451: f=   988.512, eps=0.000000000
Epoch 452: f=   988.512, eps=0.000000000
Epoch 453: f=   988.512, eps=0.000000000
Epoch 454: f=   988.512, eps=0.000000000
Epoch 455: f=   988.512, eps=0.000000000
Epoch 456: f=   988.512, eps=0.000000000
Epoch 457: f=   988.512, eps=0.000000000
Epoch 458: f=   988.512, eps=0.000000000
Epoch 459: f=   988.512, eps=0.000000000
Epoch 460: f=   988.512, eps=0.000000000
Epoch 461: f=   988.512, eps=0.000000000
Epoch 462: f=   988.512, eps=0.000000000
Epoch 463: f=   988.512, eps=0.000000000
Epoch 464: f=   988.512, eps=0.000000000
Epoch 465: f=   988.512, eps=0.000000000
Epoch 466: f=   988.512, eps=0.000000000
Epoch 467: f=   988.512, eps=0.000000000
Epoch 468: f=   988.512, eps=0.000000000
Epoch 469: f=   988.512, eps=0.000000000
Epoch 470: f=   988.512, eps=0.000000000
Epoch 471: f=   988.512, eps=0.000000000
Epoch 472: f=   988.512, eps=0.000000000
Epoch 473: f=   988.512, eps=0.000000000
Epoch 474: f=   988.512, eps=0.000000000
Epoch 475: f=   988.512, eps=0.000000000
Epoch 476: f=   988.512, eps=0.000000000
Epoch 477: f=   988.512, eps=0.000000001
Epoch 478: f=   988.512, eps=0.000000001
Epoch 479: f=   988.512, eps=0.000000001
Epoch 480: f=   988.512, eps=0.000000001
Epoch 481: f=   988.512, eps=0.000000001
```

```
Epoch 482: f=   988.512, eps=0.000000001
Epoch 483: f=   988.512, eps=0.000000001
Epoch 484: f=   988.512, eps=0.000000001
Epoch 485: f=   988.512, eps=0.000000001
Epoch 486: f=   988.512, eps=0.000000000
Epoch 487: f=   988.512, eps=0.000000000
Epoch 488: f=   988.512, eps=0.000000000
Epoch 489: f=   988.512, eps=0.000000000
Epoch 490: f=   988.512, eps=0.000000000
Epoch 491: f=   988.512, eps=0.000000000
Epoch 492: f=   988.512, eps=0.000000000
Epoch 493: f=   988.512, eps=0.000000000
Epoch 494: f=   988.512, eps=0.000000000
Epoch 495: f=   988.512, eps=0.000000000
Epoch 496: f=   988.512, eps=0.000000000
Epoch 497: f=   988.512, eps=0.000000000
Epoch 498: f=   988.512, eps=0.000000000
Epoch 499: f=   988.512, eps=0.000000000
Result after 500 epochs: f=988.511839602703
```

5.2 4b Effect of Prior

[148]:
```
import numpy as np
import pandas as pd

# Define a range of lambda values to explore
lambda_values = [0, 5, 10, 50, 100, 200, 500, 1000]
results = []

# Define columns for the results DataFrame
TRAIN_LL = 'Train LL'
TEST_LL = 'Test LL'
TEST_ACCURACY = 'Test Accuracy'
TEST_F1 = 'Test F1-Score'
LAMBDA = 'Lambda'
OBJECTIVE_VALUES = 'Objective Values'
STEP_SIZES = 'Step Sizes'
FEATURE_WEIGHTS = 'Feature Weights'


# Loop through lambda values and store results for each
for lambda_ in lambda_values:
    print(f"Training model with lambda={lambda_}...")
    try:
        # Train the model with the current lambda value
        wz_gd_l2, vz_gd_l2, ez_gd_l2 = optimize(gd_l2(y, Xz, lambda_), w0,␣
↪nepochs=500, verbose=False)
```

```
        # Calculate training log-likelihood with the current lambda
        train_log_likelihood = l_l2(y, Xz, wz_gd_l2, lambda_)

        # Calculate test log-likelihood using test data
        test_log_likelihood = l_l2(ytest, Xtestz, wz_gd_l2, lambda_)

        # Predict on the test set and calculate accuracy
        yhat_test = predict(Xtestz, wz_gd_l2)
        ypred_test = classify(Xtestz, wz_gd_l2)
        test_accuracy = sklearn.metrics.accuracy_score(ytest, ypred_test)
        test_f1 = sklearn.metrics.f1_score(ytest, ypred_test)

        # Store the results
        results.append({
            LAMBDA: lambda_,
            TRAIN_LL: train_log_likelihood,
            TEST_LL: test_log_likelihood,
            TEST_ACCURACY: test_accuracy,
            TEST_F1: test_f1,
            OBJECTIVE_VALUES: vz_gd_l2,
            STEP_SIZES: ez_gd_l2,
            FEATURE_WEIGHTS: wz_gd_l2
        })

    except Exception as e:
        print(f"Encountered an issue with lambda={lambda_}: {e}")
        results.append({
            LAMBDA: lambda_,
            TRAIN_LL: np.nan,
            TEST_LL: np.nan,
            TEST_ACCURACY: np.nan,
            TEST_F1: np.nan,
            OBJECTIVE_VALUES: None,
            STEP_SIZES: None,
            FEATURE_WEIGHTS: None
        })

# Convert results to a DataFrame for easy viewing
results_df = pd.DataFrame(results)
results_df = results_df[[LAMBDA, TRAIN_LL, TEST_LL, TEST_ACCURACY, TEST_F1]]

# Display summary statistics for each lambda
print("\nSummary of Results by Lambda:")
print(results_df)
```

```
Training model with lambda=0…
```

```
Training model with lambda=5…
Training model with lambda=10…
Training model with lambda=50…
Training model with lambda=100…
Training model with lambda=200…
Training model with lambda=500…
Training model with lambda=1000…
```

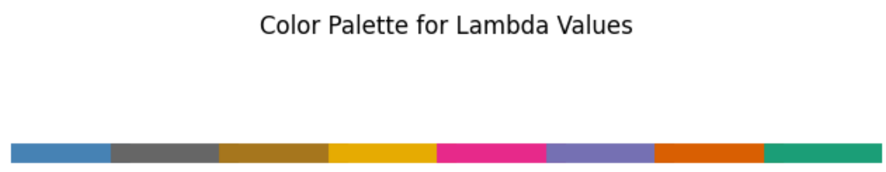
```
/var/folders/t3/h38q5w_d36ncdxty42rj79mr0000gn/T/ipykernel_48153/1382573211.py:4
: RuntimeWarning: overflow encountered in exp
  return 1 / (1 + np.exp(-x))
/var/folders/t3/h38q5w_d36ncdxty42rj79mr0000gn/T/ipykernel_48153/2230293396.py:4
: RuntimeWarning: divide by zero encountered in log
  return np.log(sigma(x))
/var/folders/t3/h38q5w_d36ncdxty42rj79mr0000gn/T/ipykernel_48153/3910309765.py:1
3: RuntimeWarning: invalid value encountered in multiply
  log_likelihood = np.sum(y * logsigma( X @ w) + (1-y) * logsigma(- X @ w))
/var/folders/t3/h38q5w_d36ncdxty42rj79mr0000gn/T/ipykernel_48153/2045031579.py:9
: RuntimeWarning: overflow encountered in multiply
  gradient_l2 = lambda_ * w
/var/folders/t3/h38q5w_d36ncdxty42rj79mr0000gn/T/ipykernel_48153/3910309765.py:1
3: RuntimeWarning: invalid value encountered in matmul
  log_likelihood = np.sum(y * logsigma( X @ w) + (1-y) * logsigma(- X @ w))
/var/folders/t3/h38q5w_d36ncdxty42rj79mr0000gn/T/ipykernel_48153/577217326.py:19
: RuntimeWarning: invalid value encountered in matmul
  e = y - sigma(X @ w)
```

```
Summary of Results by Lambda:
   Lambda     Train LL     Test LL  Test Accuracy  Test F1-Score
0       0  -655.413496 -427.344200       0.918620       0.893436
1       5  -722.372059 -473.222438       0.919922       0.894962
2      10  -754.852420 -492.753731       0.919922       0.895141
3      50  -893.111219 -575.721059       0.917969       0.893220
4     100  -988.511840 -632.242151       0.916667       0.891892
5     200 -1108.946040 -701.367391       0.911458       0.885714
6     500 -1304.383410 -805.992635       0.907552       0.881271
7    1000          NaN         NaN       0.612630       0.000000
```

Color Palette for Lambda Values

[149]:
```
# Colors for each lambda, with the first color set to GD_MLE_COLOR
colors = [GD_MLE_COLOR] + list(plt.cm.Dark2_r(np.linspace(0, 1,␣
 ↪len(lambda_values) - 1)))

# plt.cm.Dark2_r

# Display the colors as a palette
plt.figure(figsize=(8, 2))
```

```
for i, color in enumerate(colors):
    plt.plot([i, i+1], [1, 1], lw=10, color=color)

plt.xlim(0, len(colors))
plt.ylim(0.5, 1.5)
plt.axis('off')
plt.title("Color Palette for Lambda Values")
plt.show()
```



5.2.1 Resulting dataframe

[150]:
```
import matplotlib.pyplot as plt
from pandas.plotting import table
import matplotlib.colors as mcolors

# Round results for better readability
results_df = results_df.round(2)

# Define colors for highlighting
highlight_green = mcolors.to_rgba('lightgreen', alpha=0.5)
highlight_red = mcolors.to_rgba('lightcoral', alpha=0.5)

# Set the size of the figure
fig, ax = plt.subplots(figsize=(8, 4))  # Adjust figsize as needed

# Hide axes
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
ax.set_frame_on(False)

# Create the table and add it to the figure
tbl = table(ax, results_df, loc='center', cellLoc='center', colWidths=[0.15] *␣
↪len(results_df.columns))

# Format the table
tbl.auto_set_font_size(False)
```

```python
tbl.set_fontsize(10)
tbl.scale(1.2, 1.2)

# Set the colors for the "Lambda" column based on predefined colors list if␣
↪provided
for i, color in enumerate(colors):
    cell = tbl[(i + 1, 0)]  # (i+1, 0) is the (row, col) position in the table␣
↪for Lambda column
    cell.set_text_props(color="white")  # Set text color to white for␣
↪readability
    cell.set_facecolor(color)           # Set background color to the␣
↪corresponding color

# Highlight the max values in the "Test Accuracy" and "Test F1-Score" columns
max_accuracy = results_df[TEST_ACCURACY].max()
for i, accuracy in enumerate(results_df[TEST_ACCURACY]):
    cell = tbl[(i + 1, 3)]  # (i+1, 3) is the (row, col) position in the table␣
↪for Test Accuracy column
    if accuracy == max_accuracy:
        cell.set_facecolor(highlight_green)  # Light green for max value
    else:
        cell.set_facecolor(highlight_red)    # Light red for others


max_f1_score = results_df[TEST_F1].max()
for i, f1_score in enumerate(results_df[TEST_F1]):
    cell = tbl[(i + 1, 4)]  # (i+1, 4) is the (row, col) position in the table␣
↪for Test F1-Score column
    if f1_score == max_f1_score:
        cell.set_facecolor(highlight_green)  # Light green for max value
    else:
        cell.set_facecolor(highlight_red)    # Light red for others

# Display the plot
plt.show()
```

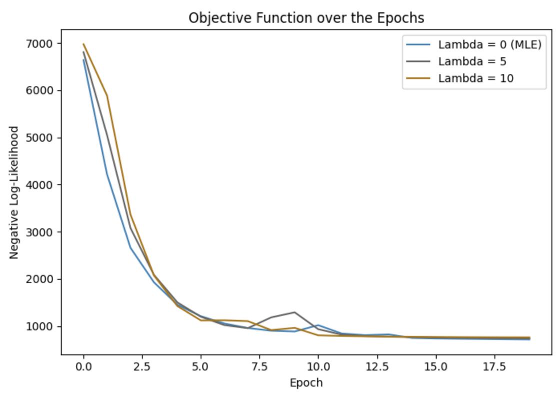
	Lambda	Train LL	Test LL	Test Accuracy	Test F1-Score
0	0.0	-655.41	-427.34	0.92	0.89
1	5.0	-722.37	-473.22	0.92	0.89
2	10.0	-754.85	-492.75	0.92	0.9
3	50.0	-893.11	-575.72	0.92	0.89
4	100.0	-988.51	-632.24	0.92	0.89
5	200.0	-1108.95	-701.37	0.91	0.89
6	500.0	-1304.38	-805.99	0.91	0.88
7	1000.0	nan	nan	0.61	0.0

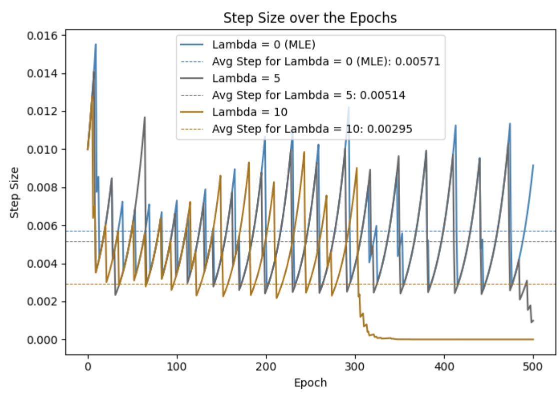
5.2.2 Objective function and step size across epochs for each value

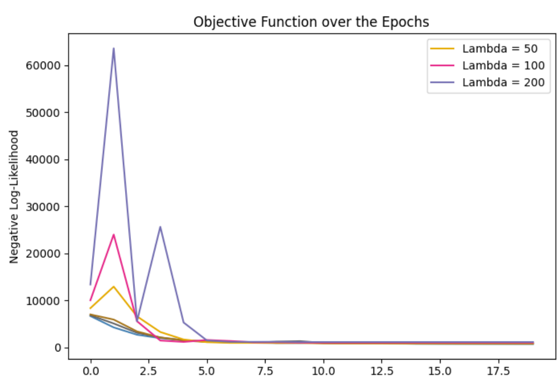
[151]:
```
import matplotlib.pyplot as plt
import numpy as np

def plot_results(results, colors, filter_lambdas=None, highlight_lambdas=None):
    """
    Plot objective function and step size over epochs for given results, with␣
 ↪optional filtering
    and specific lambda highlights.

    Parameters:
        results (list): List of dictionaries with keys for lambda, objective␣
 ↪values, and step sizes.
        colors (list): List of color codes to use for each lambda plot.
        filter_lambdas (float): Upper limit for lambda values to include in␣
 ↪plots (None for no limit).
        highlight_lambdas (list): Specific lambda values to highlight with␣
 ↪labels in the plots.
    """

    # Define figure with two side-by-side subplots
    plt.figure(figsize=(14, 5))

    # 1st Plot: Objective Function over Epochs
    plt.subplot(1, 2, 1)
    for i, result in enumerate(results):
        lambda_ = result[LAMBDA]
```

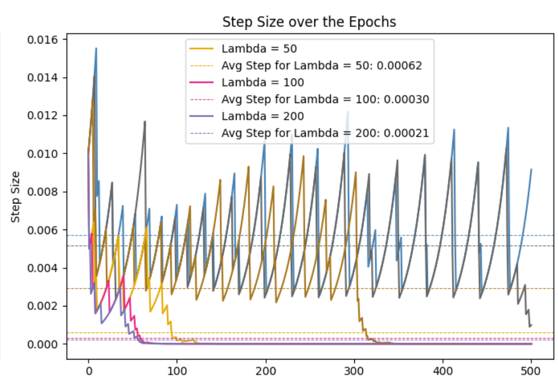
```python
        objective_values = result[OBJECTIVE_VALUES]

        # Apply filter based on lambda value
        if filter_lambdas is None or lambda_ <= filter_lambdas:
            label = f"Lambda = {lambda_} (MLE)" if lambda_ == 0 else f"Lambda =␣
↪{lambda_}"
            # Highlight label for specific lambdas, otherwise no label
            label = label if highlight_lambdas is None or lambda_ in␣
↪highlight_lambdas else None
            if objective_values is not None:
                plt.plot(objective_values[:20], label=label, color=colors[i])
    plt.title("Objective Function over the Epochs")
    plt.xlabel("Epoch")
    plt.ylabel("Negative Log-Likelihood")
    if highlight_lambdas:
        plt.legend()

    # 2nd Plot: Step Size over Epochs
    plt.subplot(1, 2, 2)
    for i, result in enumerate(results):
        lambda_ = result[LAMBDA]
        step_sizes = result[STEP_SIZES]

        # Apply filter based on lambda value
        if filter_lambdas is None or lambda_ <= filter_lambdas:
            label = f"Lambda = {lambda_} (MLE)" if lambda_ == 0 else f"Lambda =␣
↪{lambda_}"
            # Highlight label for specific lambdas, otherwise no label
            label = label if highlight_lambdas is None or lambda_ in␣
↪highlight_lambdas else None
            if step_sizes is not None:
                plt.plot(step_sizes, label=label, color=colors[i])
                plt.axhline(np.mean(step_sizes), linestyle='--', lw=0.7,␣
↪color=colors[i],
                            label=(f'Avg Step for {label}: {np.mean(step_sizes):
↪.5f}' if label else None))
    plt.title("Step Size over the Epochs")
    plt.xlabel("Epoch")
    plt.ylabel("Step Size")
    if highlight_lambdas:
        plt.legend()

    # Adjust layout and display
    plt.tight_layout()
    plt.show()
```

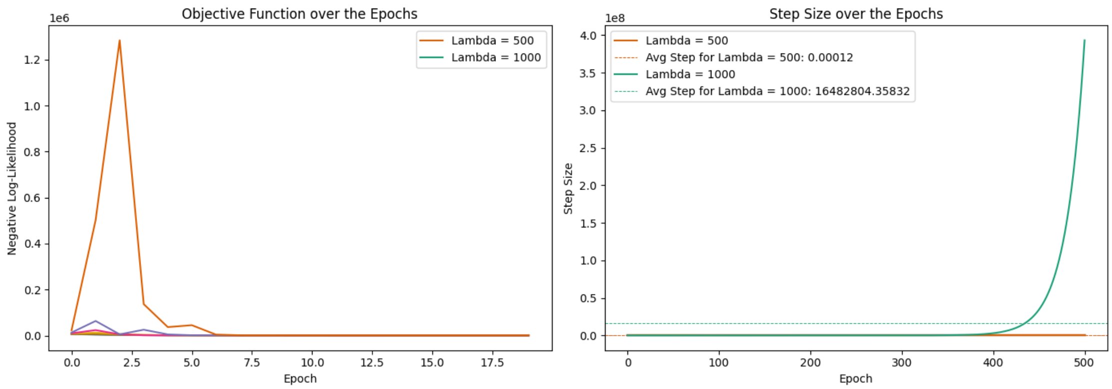
```python
# Extract unique lambdas from results dynamically
all_lambdas = sorted({result[LAMBDA] for result in results})

# Lists to hold the upper limits and dynamic labels for each plot
filter_limits = [10, 200, max(all_lambdas)]
previous_lambdas = set()

# Loop over each filter limit and dynamically determine highlight_lambdas
for limit in filter_limits:
    current_lambdas = {l for l in all_lambdas if l <= limit}
    # Compute new lambdas that are not in previous plots
    highlight_lambdas = sorted(current_lambdas - previous_lambdas)

    # Plot with the current filter and highlight
    plot_results(results, colors, filter_lambdas=limit, highlight_lambdas=highlight_lambdas)

    # Update previous lambdas for the next iteration
    previous_lambdas.update(current_lambdas)
```







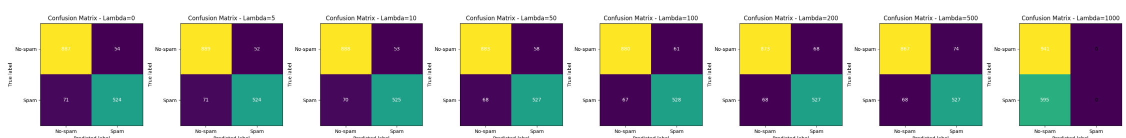




5.2.3 Confusion Matrices

[152]:
```
import matplotlib.pyplot as plt
import numpy as np
import sklearn.metrics

fig, axes = plt.subplots(1, len(lambda_values), figsize=(4 *␣
↪len(lambda_values), 4))

# Define tick labels
tick_labels = ['No-spam', 'Spam']
ticks = np.arange(len(tick_labels))

for idx, result in enumerate(results):
    lambda_value = result[LAMBDA]
    weights = result[FEATURE_WEIGHTS]

    if weights is not None:
        # Predict and classify with the weights for the current lambda value
        yhat = predict(Xtestz, weights)
        ypred = classify(Xtestz, weights)

        # Generate confusion matrix
        confusion_matrix = sklearn.metrics.confusion_matrix(ytest, ypred)

        # Plot the confusion matrix for the current lambda value
        ax = axes[idx]
        ax.imshow(confusion_matrix, interpolation='nearest', cmap=plt.cm.
↪viridis)
        ax.set_title(f'Confusion Matrix - Lambda={lambda_value}')
```

```
        ax.set_xlabel('Predicted label')
        ax.set_ylabel('True label')

        # Set tick labels
        ax.set_xticks(ticks)
        ax.set_yticks(ticks)
        ax.set_xticklabels(tick_labels)
        ax.set_yticklabels(tick_labels)

        # Add text annotations to each cell
        for i in range(2):
            for j in range(2):
                ax.text(j, i, confusion_matrix[i, j], ha='center', va='center',
                        color='white' if confusion_matrix[i, j] > 50 else␣
↪'black')
    else:
        # If there was an issue training this model, add a placeholder
        ax = axes[idx]
        ax.text(0.5, 0.5, 'N/A', ha='center', va='center', fontsize=12,␣
↪color='red')
        ax.set_title(f'Confusion Matrix - Lambda={lambda_value}')
        ax.set_xlabel('Predicted label')
        ax.set_ylabel('True label')
        ax.set_xticks([])
        ax.set_yticks([])

plt.tight_layout()
plt.show()
```

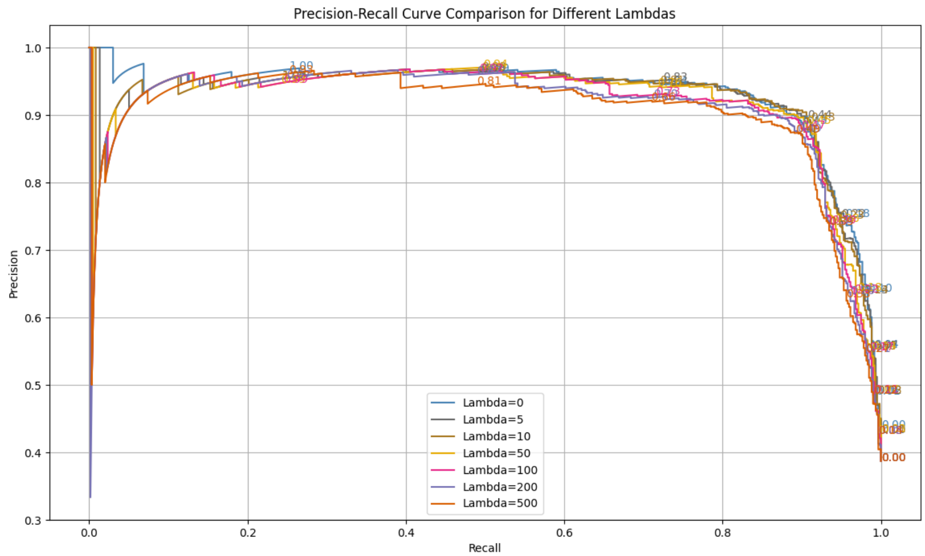


5.2.4 Precision-Recall Curve Comparison for Different Lambdas

[153]:
```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.metrics as metrics

# Plot Precision-Recall curves for each lambda
plt.figure(figsize=(14, 8))

for idx, result in enumerate(results):
    lambda_value = result[LAMBDA]
```

```
    weights = result[FEATURE_WEIGHTS]

    if weights is not None:
        # Predict probabilities and calculate precision-recall metrics
        yhat = predict(Xtestz, weights)

        try:
            # Attempt to calculate precision-recall curve
            precision, recall, thresholds = metrics.
 ↪precision_recall_curve(ytest, yhat)

            # Plot Precision-Recall curve
            plt.plot(recall, precision, label=f"Lambda={lambda_value}",␣
 ↪color=colors[idx])

            # Add threshold values at selected points along the curve
            for x in np.linspace(0, 1, 10, endpoint=False):
                index = int(x * (precision.size - 1))
                plt.text(recall[index], precision[index], f"{thresholds[index]:.
 ↪2f}", color=colors[idx])

        except ValueError as e:
            print(f"Skipping lambda={lambda_value} due to NaN in predictions or␣
 ↪other errors.")
            continue  # Skip this lambda if precision-recall calculation fails

# Labels and title
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve Comparison for Different Lambdas")
plt.legend()
plt.grid(True)
plt.show()
```

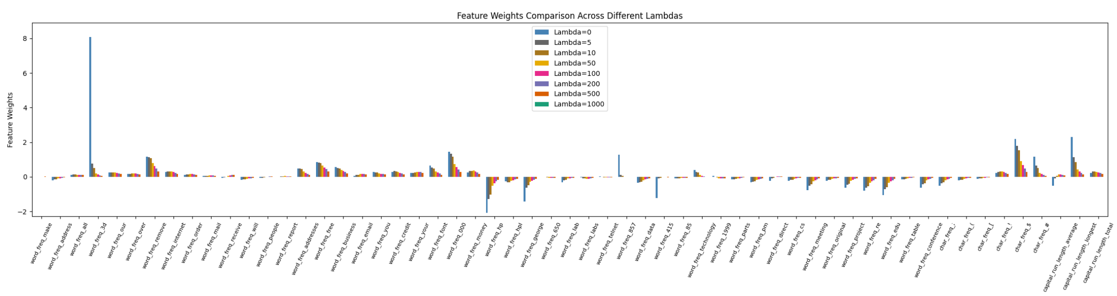
```
Skipping lambda=1000 due to NaN in predictions or other errors.
```



5.3 4c Composition of Weight Vector

[154]:
```
import matplotlib.pyplot as plt
import numpy as np

# Set up the figure for a comparison across lambda values
fig, ax = plt.subplots(figsize=(len(features) * 0.4, 6))
x_positions = np.arange(len(features))

# Plot weights for each lambda value
bar_width = 0.8 / len(lambda_values)  # Width of each bar to fit all lambdas␣
↪side-by-side
for idx, result in enumerate(results):
    lambda_value = result[LAMBDA]
    feature_weights = result[FEATURE_WEIGHTS]

    if feature_weights is not None:
        # Offset the bars for each lambda value
        ax.bar(x_positions + (idx - len(lambda_values) / 2) * bar_width,␣
↪feature_weights,
               width=bar_width, color=colors[idx],␣
↪label=f"Lambda={lambda_value}")

# Set feature names on the x-axis as ticks
ax.set_xticks(x_positions)
ax.set_xticklabels(features, rotation=65, fontsize=8)
```

```python
ax.set_xlim(-0.5, len(features) - 0.5)

# Add labels and title
ax.set_ylabel("Feature Weights")
plt.title("Feature Weights Comparison Across Different Lambdas")

# Determine the range for the y-axis to include all weight values
all_weights = [w for result in results if result[FEATURE_WEIGHTS] is not None for w in result[FEATURE_WEIGHTS]]
min_weight = min(all_weights)
max_weight = max(all_weights)
ax.set_ylim([min_weight * 1.1, max_weight * 1.1])

# Add a legend and tighten layout
ax.legend()
plt.tight_layout()

# Display the plot
plt.show()
```



5.4 5 Exploration (optional)

5.4.1 5 Exploration: PyTorch

[155]:
```python
# if you want to experiment, here is an implementation of logistic
# regression in PyTorch
import math
import torch
import torch.nn as nn
import torch.utils.data
import torch.nn.functional as F

# prepare the data
Xztorch = torch.FloatTensor(Xz)
ytorch = torch.LongTensor(y)
train = torch.utils.data.TensorDataset(Xztorch, ytorch)
```

```python
# manual implementation of logistic regression (without bias)
class LogisticRegression(nn.Module):
    def __init__(self, D, C):
        super(LogisticRegression, self).__init__()
        self.weights = torch.nn.Parameter(
            torch.randn(D, C) / math.sqrt(D)
        )  # xavier initialization
        self.register_parameter("W", self.weights)

    def forward(self, x):
        out = torch.matmul(x, self.weights)
        out = F.log_softmax(out)
        return out


# define the objective and update function. here we ignore the learning rates
# and parameters given to us by optimize (they are stored in the PyTorch model
# and optimizer, resp., instead)
def opt_pytorch():
    model = LogisticRegression(D, 2)
    criterion = nn.NLLLoss(reduction="sum")
    # change the next line to try different optimizers
    # optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    def objective(_):
        outputs = model(Xztorch)
        return criterion(outputs, ytorch)

    def update(_1, _2):
        for i, (examples, labels) in enumerate(train_loader):
            outputs = model(examples)
            loss = criterion(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        W = model.state_dict()["W"]
        w = W[:, 1] - W[:, 0]
        return w

    return (objective, update)
```

[156]:

```python
# run the optimizer
learning_rate = 0.01
```

```
batch_size = 100  # number of data points to sample for gradient estimate
shuffle = True  # sample with replacement (false) or without replacement (true)

train_loader = torch.utils.data.DataLoader(train, batch_size, shuffle=True)
wz_t, vz_t, _ = optimize(opt_pytorch(), None, nepochs=100, eps0=None,␣
 ↪verbose=True)
```

```
/var/folders/t3/h38q5w_d36ncdxty42rj79mr0000gn/T/ipykernel_48153/2194961090.py:2
6: UserWarning: Implicit dimension choice for log_softmax has been deprecated.
Change the call to include dim=X as an argument.
  out = F.log_softmax(out)
```

```
Epoch   0: f=  2401.555, eps=       nan
Epoch   1: f=   852.813, eps=       nan
Epoch   2: f=   769.840, eps=       nan
Epoch   3: f=   734.660, eps=       nan
Epoch   4: f=   715.665, eps=       nan
Epoch   5: f=   703.678, eps=       nan
Epoch   6: f=   694.922, eps=       nan
Epoch   7: f=   689.826, eps=       nan
Epoch   8: f=   685.402, eps=       nan
Epoch   9: f=   681.408, eps=       nan
Epoch  10: f=   677.918, eps=       nan
Epoch  11: f=   676.084, eps=       nan
Epoch  12: f=   674.582, eps=       nan
Epoch  13: f=   672.422, eps=       nan
Epoch  14: f=   671.319, eps=       nan
Epoch  15: f=   669.324, eps=       nan
Epoch  16: f=   669.503, eps=       nan
Epoch  17: f=   668.074, eps=       nan
Epoch  18: f=   667.481, eps=       nan
Epoch  19: f=   666.695, eps=       nan
Epoch  20: f=   665.682, eps=       nan
Epoch  21: f=   665.047, eps=       nan
Epoch  22: f=   664.510, eps=       nan
Epoch  23: f=   663.521, eps=       nan
Epoch  24: f=   662.501, eps=       nan
Epoch  25: f=   662.371, eps=       nan
Epoch  26: f=   662.070, eps=       nan
Epoch  27: f=   660.921, eps=       nan
Epoch  28: f=   659.974, eps=       nan
Epoch  29: f=   660.324, eps=       nan
Epoch  30: f=   660.363, eps=       nan
Epoch  31: f=   658.990, eps=       nan
Epoch  32: f=   658.953, eps=       nan
Epoch  33: f=   658.677, eps=       nan
Epoch  34: f=   657.691, eps=       nan
Epoch  35: f=   658.102, eps=       nan
```

```
Epoch  36: f=   657.104, eps=        nan
Epoch  37: f=   656.706, eps=        nan
Epoch  38: f=   656.186, eps=        nan
Epoch  39: f=   656.916, eps=        nan
Epoch  40: f=   656.110, eps=        nan
Epoch  41: f=   655.836, eps=        nan
Epoch  42: f=   655.266, eps=        nan
Epoch  43: f=   654.581, eps=        nan
Epoch  44: f=   653.945, eps=        nan
Epoch  45: f=   653.735, eps=        nan
Epoch  46: f=   653.343, eps=        nan
Epoch  47: f=   652.729, eps=        nan
Epoch  48: f=   652.832, eps=        nan
Epoch  49: f=   652.829, eps=        nan
Epoch  50: f=   652.807, eps=        nan
Epoch  51: f=   651.720, eps=        nan
Epoch  52: f=   653.507, eps=        nan
Epoch  53: f=   655.165, eps=        nan
Epoch  54: f=   652.750, eps=        nan
Epoch  55: f=   651.120, eps=        nan
Epoch  56: f=   651.349, eps=        nan
Epoch  57: f=   650.208, eps=        nan
Epoch  58: f=   649.970, eps=        nan
Epoch  59: f=   650.703, eps=        nan
Epoch  60: f=   649.896, eps=        nan
Epoch  61: f=   649.680, eps=        nan
Epoch  62: f=   649.484, eps=        nan
Epoch  63: f=   648.847, eps=        nan
Epoch  64: f=   651.523, eps=        nan
Epoch  65: f=   649.589, eps=        nan
Epoch  66: f=   648.478, eps=        nan
Epoch  67: f=   647.883, eps=        nan
Epoch  68: f=   647.869, eps=        nan
Epoch  69: f=   647.849, eps=        nan
Epoch  70: f=   647.210, eps=        nan
Epoch  71: f=   648.278, eps=        nan
Epoch  72: f=   647.711, eps=        nan
Epoch  73: f=   646.818, eps=        nan
Epoch  74: f=   646.698, eps=        nan
Epoch  75: f=   647.395, eps=        nan
Epoch  76: f=   645.831, eps=        nan
Epoch  77: f=   645.746, eps=        nan
Epoch  78: f=   645.454, eps=        nan
Epoch  79: f=   646.279, eps=        nan
Epoch  80: f=   645.348, eps=        nan
Epoch  81: f=   645.426, eps=        nan
Epoch  82: f=   644.814, eps=        nan
Epoch  83: f=   645.585, eps=        nan
```

```
Epoch  84: f=   644.798, eps=       nan
Epoch  85: f=   645.399, eps=       nan
Epoch  86: f=   644.990, eps=       nan
Epoch  87: f=   644.231, eps=       nan
Epoch  88: f=   644.118, eps=       nan
Epoch  89: f=   644.396, eps=       nan
Epoch  90: f=   643.846, eps=       nan
Epoch  91: f=   643.350, eps=       nan
Epoch  92: f=   643.066, eps=       nan
Epoch  93: f=   643.079, eps=       nan
Epoch  94: f=   642.726, eps=       nan
Epoch  95: f=   642.814, eps=       nan
Epoch  96: f=   642.965, eps=       nan
Epoch  97: f=   643.182, eps=       nan
Epoch  98: f=   642.654, eps=       nan
Epoch  99: f=   643.001, eps=       nan
Result after 100 epochs: f=642.265625
```