lab_gene_partial (2)

October 15, 2023

Name: Shadeeb Hossain

ID:sh7492

1 Lab: Logistic Regression for Gene Expression Data

In this lab, we use logistic regression to predict biological characteristics ("phenotypes") from gene expression data. In addition to the concepts in breast cancer demo, you will learn to: * Handle missing data * Perform multi-class logistic classification * Create a confusion matrix * Use L1-regularization for improved estimation in the case of sparse weights (Grad students only)

1.1 Background

Genes are the basic unit in the DNA and encode blueprints for proteins. When proteins are synthesized from a gene, the gene is said to "express". Micro-arrays are devices that measure the expression levels of large numbers of genes in parallel. By finding correlations between expression levels and phenotypes, scientists can identify possible genetic markers for biological characteristics.

The data in this lab comes from:

https://archive.ics.uci.edu/ml/datasets/Mice+Protein+Expression

In this data, mice were characterized by three properties: * Whether they had down's syndrome (trisomy) or not * Whether they were stimulated to learn or not * Whether they had a drug memantine or a saline control solution.

With these three choices, there are 8 possible classes for each mouse. For each mouse, the expression levels were measured across 77 genes. We will see if the characteristics can be predicted from the gene expression levels. This classification could reveal which genes are potentially involved in Down's syndrome and if drugs and learning have any noticeable effects.

1.2 Load the Data

We begin by loading the standard modules.

[85]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

```
from sklearn import linear_model, preprocessing
```

Use the `pd.read_excel` command to read the data from

https://archive.ics.uci.edu/ml/machine-learning-databases/00342/Data_Cortex_Nuclear.xls

into a dataframe `df`. Use the `index_col` option to specify that column 0 is the index. Use the `df.head()` to print the first few rows.

[86]:
```
# TODO 1
import pandas as pd
df = pd.read_excel("Data_Cortex_Nuclear.xls",index_col=0)
df.head(160)
```

[86]:
```
         DYRK1A_N   ITSN1_N    BDNF_N     NR1_N    NR2A_N    pAKT_N   pBRAF_N  \
MouseID
309_1    0.503644  0.747193  0.430175  2.816329  5.990152  0.218830  0.177565
309_2    0.514617  0.689064  0.411770  2.789514  5.685038  0.211636  0.172817
309_3    0.509183  0.730247  0.418309  2.687201  5.622059  0.209011  0.175722
309_4    0.442107  0.617076  0.358626  2.466947  4.979503  0.222886  0.176463
309_5    0.434940  0.617430  0.358802  2.365785  4.718679  0.213106  0.173627
…             …         …         …         …         …         …         …
294_6    0.309966  0.468068  0.283041  2.415695  4.080775  0.232638  0.171236
294_7    0.323467  0.478436  0.316702  2.558985  3.715011  0.245455  0.201903
294_8    0.293755  0.462408  0.294577  2.421323  3.884141  0.254519  0.192276
294_9    0.280736  0.434656  0.281317  2.299322  3.722556  0.266215  0.179090
294_10   0.339310  0.521326  0.343385  2.550937  3.437109  0.271937  0.220321

         pCAMKII_N   pCREB_N    pELK_N  …   pCFOS_N     SYP_N  H3AcK18_N  \
MouseID                                 …
309_1     2.373744  0.232224  1.750936  …  0.108336  0.427099   0.114783
309_2     2.292150  0.226972  1.596377  …  0.104315  0.441581   0.111974
309_3     2.283337  0.230247  1.561316  …  0.106219  0.435777   0.111883
309_4     2.152301  0.207004  1.595086  …  0.111262  0.391691   0.130405
309_5     2.134014  0.192158  1.504230  …  0.110694  0.434154   0.118481
…              …         …         …  …  …         …          …
294_6     5.105073  0.204072  1.290921  …  0.120253  0.483252        NaN
294_7     5.254123  0.241226  1.368076  …  0.145763  0.469362        NaN
294_8     5.088537  0.225349  1.349425  …  0.127904  0.469476        NaN
294_9     4.840465  0.198064  1.240465  …  0.135542  0.477912        NaN
294_10    5.197501  0.239337  1.499864  …  0.157727  0.481776        NaN

           EGR1_N  H3MeK4_N    CaNA_N  Genotype  Treatment  Behavior   class
MouseID
309_1    0.131790  0.128186  1.675652   Control  Memantine       C/S  c-CS-m
309_2    0.135103  0.131119  1.743610   Control  Memantine       C/S  c-CS-m
309_3    0.133362  0.127431  1.926427   Control  Memantine       C/S  c-CS-m
309_4    0.147444  0.146901  1.700563   Control  Memantine       C/S  c-CS-m
309_5    0.140314  0.148380  1.839730   Control  Memantine       C/S  c-CS-m
```

```
...         ...       ...       ...       ...        ...        ...      ...
294_6     0.212171  0.187926  1.051022   Control  Memantine        S/C  c-SC-m
294_7     0.238737  0.238487  1.075877   Control  Memantine        S/C  c-SC-m
294_8     0.237148  0.228868  1.041646   Control  Memantine        S/C  c-SC-m
294_9     0.235191  0.216240  1.082706   Control  Memantine        S/C  c-SC-m
294_10    0.250676  0.243673  1.076874   Control  Memantine        S/C  c-SC-m

[160 rows x 81 columns]
```

This data has missing values. The site:

http://pandas.pydata.org/pandas-docs/stable/missing_data.html

has an excellent summary of methods to deal with missing values. Following the techniques there, create a new data frame `df1` where the missing values in each column are filled with the mean values from the non-missing values.

[87]:
```python
# TODO 2
df1 =df.fillna(df.mean())
```

```
<ipython-input-87-063fb173b661>:2: FutureWarning: The default value of
numeric_only in DataFrame.mean is deprecated. In a future version, it will
default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence this
warning.
  df1 =df.fillna(df.mean())
```

1.3 Binary Classification for Down's Syndrome

We will first predict the binary class label in `df1['Genotype']` which indicates if the mouse has Down's syndrome or not. Get the string values in `df1['Genotype'].values` and convert this to a numeric vector `y` with 0 or 1. You may wish to use the `np.unique` command with the `return_inverse=True` option.

[88]:
```python
# TODO 3
genotype_values = df1['Genotype'].values
y, _ = np.unique(genotype_values, return_inverse=True)
print(y)

yraw = np.array(df['Genotype'])
BEN_VAL = 'Control' # value in the 'class' label for benign samples
MAL_VAL = 'Ts65Dn'    # value in the 'class' label for malignant samples
y = (yraw == MAL_VAL).astype(str)
Iben = (y==0)
Imal = (y==1)
```

```
['Control' 'Ts65Dn']
```

```
<ipython-input-88-2217ede8e488>:10: FutureWarning: elementwise comparison
failed; returning scalar instead, but in the future will perform elementwise
comparison
```

```
  Iben = (y==0)
<ipython-input-88-2217ede8e488>:11: FutureWarning: elementwise comparison
failed; returning scalar instead, but in the future will perform elementwise
comparison
  Imal = (y==1)
```

As predictors, get all but the last four columns of the dataframes. Store the data matrix into `X` and the names of the columns in `xnames`.

[89]:
```
# TODO 4
X = df1.iloc[:, :-4]
xnames = X.columns
```

Split the data into training and test with 30% allocated for test. You can use the train

[90]:
```
#from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
Xtr, Xts, ytr, yts = train_test_split(X,y, test_size=0.30)
```

Scale the data with the `StandardScaler`. Store the scaled values in `Xtr1` and `Xts1`.

[91]:
```
#from sklearn.preprocessing import
from sklearn.preprocessing import StandardScaler

scal = StandardScaler()
Xtr1 = scal.fit_transform(Xtr)
Xts1 = scal.transform(Xts)
```

Create a `LogisticRegression` object `logreg` and `fit` on the scaled training data. Set the regularization level to `C=1e5` and use the optimizer `solver=liblinear`.

[92]:
```
# TODO 7
logreg = linear_model.LogisticRegression(C=1e5,solver='liblinear')
logreg.fit(Xtr1, ytr)
#   logreg = ...
```

[92]:
```
LogisticRegression(C=100000.0, solver='liblinear')
```

Measure the accuracy of the classifer on test data. You should get around 94%.

[93]:
```
# TODO 8
yhat = logreg.predict(Xts1)
acc = np.mean(yhat == yts)
print("Accuracy on test data = %f" % acc)
```

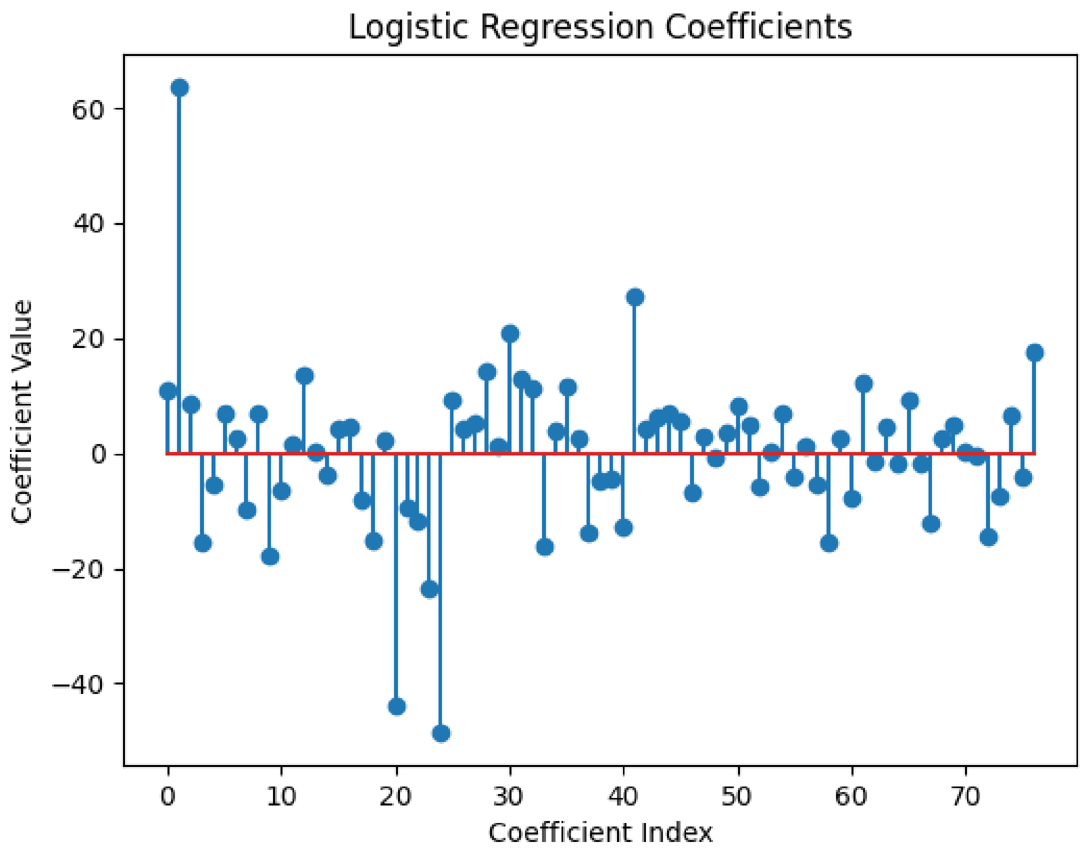
```
Accuracy on test data = 0.959877
```

1.4 Interpreting the weight vector

Create a stem plot of the coefficients, `W` in the logistic regression model. Jse the `plt.stem()` function with the `use_line_collection=True` option. You can get the coefficients from `logreg.coef_`, but you will need to reshape this to a 1D array.

[94]:
```python
# TODO 9
W= logreg.coef_.ravel()

# Create an array of indices for the coefficients
indices = np.arange(len(W))

# Create the stem plot
plt.stem(indices, W, use_line_collection=True)

# Set labels and title
plt.xlabel('Coefficient Index')
plt.ylabel('Coefficient Value')
plt.title('Logistic Regression Coefficients')

# Show the plot
plt.show()
#   W = ...
#   plt.stem(...)
```

```
<ipython-input-94-0be2abc58560>:8: MatplotlibDeprecationWarning: The
'use_line_collection' parameter of stem() was deprecated in Matplotlib 3.6 and
will be removed two minor releases later. If any parameter follows
'use_line_collection', they should be passed as keyword, not positionally.
  plt.stem(indices, W, use_line_collection=True)
```



You should see that `W[i]` is very large for a few components `i`. These are the genes that are likely to be most involved in Down's Syndrome. Below we will use L1 regression to enforce sparsity. Find the names of the genes for two components `i` where the magnitude of `W[i]` is largest.

[100]:
```
# TODO 10
W = logreg.coef_
largest_indices = (-np.abs(W)).argsort(axis=1)[:, :2]
gene_names = [xnames[i] for i in largest_indices.ravel()]
print("Gene names for two components with the largest magnitude:")
for i, gene_name in enumerate(gene_names):
    print(f"Component {i + 1}: {gene_name}")
```

```
Gene names for two components with the largest magnitude:
Component 1: ITSN1_N
Component 2: ERK_N
```

1.5 Cross Validation

To obtain a slightly more accurate result, now perform 10-fold cross validation and measure the average precision, recall and f1-score. Note, that in performing the cross-validation, you will want to randomly permute the test and training sets using the `shuffle` option. In this data set, all the samples from each class are bunched together, so shuffling is essential. Print the mean precision, recall and f1-score and error rate across all the folds.

[96]:
```
#TODO 11

from sklearn.model_selection import KFold
from sklearn.metrics import precision_recall_fscore_support
nfold = 10
kf = KFold(n_splits=nfold,shuffle=True)

acc = np.zeros(nfold)
prec = np.zeros(nfold)
rec = np.zeros(nfold)
f1 = np.zeros(nfold)

for i, I in enumerate(kf.split(X)):

    # Get training and test data
    train, test = I
    Xtr11 = X[train:]
    ytr11 = y[train]
    Xts11 = X[test,:]
    yts11 = y[test]

    # Scale the data
    scal = StandardScaler()
    Xtr1 = scal.fit_transform(Xtr11)
    Xts1 = scal.transform(Xts11)

    # Fit a model
    logreg.fit(Xtr1, ytr11)

    # Predict on test samples and measure accuracy
    yhat = logreg.predict(Xts1)
    acc[i] = np.mean(yhat == yts11)

    # Measure other performance metrics
    prec[i],rec[i],f1[i],_  =␣
 ↪precision_recall_fscore_support(yts11,yhat,average='binary')


# Take average values of the metrics
precm = np.mean(prec)
recm = np.mean(rec)
f1m = np.mean(f1)
accm= np.mean(acc)
```

```
# Compute the standard errors
prec_se = np.std(prec)/np.sqrt(nfold-1)
rec_se = np.std(rec)/np.sqrt(nfold-1)
f1_se = np.std(f1)/np.sqrt(nfold-1)
acc_se = np.std(acc)/np.sqrt(nfold-1)

print('Precision = {0:.4f}, SE={1:.4f}'.format(precm,prec_se))
print('Recall =    {0:.4f}, SE={1:.4f}'.format(recm, rec_se))
print('f1 =        {0:.4f}, SE={1:.4f}'.format(f1m, f1_se))
print('Accuracy =  {0:.4f}, SE={1:.4f}'.format(accm, acc_se))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪get_loc(self, key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

/usr/local/lib/python3.10/dist-packages/pandas/_libs/index.pyx in pandas._libs.
 ↪index.IndexEngine.get_loc()

/usr/local/lib/python3.10/dist-packages/pandas/_libs/index.pyx in pandas._libs.
 ↪index.IndexEngine.get_loc()

TypeError: '[   1    2    3    4    5    6    7    8    9   10   11   12   13
 ↪14
   15   16   17   18   20   21   22   23   24   25   26   27   28   29
   30   31   32   33   34   35   36   38   39   40   42   43   44   45
   46   47   48   49   50   51   52   53   54   55   56   59   60   62
   63   64   65   66   67   68   69   70   71   72   73   74   75   76
   77   78   79   80   81   82   84   85   86   87   89   90   91   92
   93   94   95   96   97   98   99  101  102  103  104  105  106  107
  108  109  110  111  112  113  114  115  117  119  121  122  123  124
  125  126  128  130  131  132  133  134  135  136  137  138  139  140
  141  142  143  144  145  146  148  149  150  151  152  153  154  155
  156  157  158  159  161  164  165  166  167  168  169  170  171  172
  173  174  175  176  177  178  179  180  181  182  183  184  185  186
  187  188  189  190  191  192  193  194  195  196  197  198  199  200
  201  202  203  204  205  206  207  208  209  210  211  213  215  216
  217  218  219  220  221  222  223  224  225  226  227  229  231  232
  233  234  235  236  237  238  239  240  241  242  243  245  246  247
  248  249  250  251  252  253  254  255  256  257  258  259  260  261
  263  264  265  266  269  270  272  273  274  275  276  277  278  279
  280  281  282  283  284  285  286  288  289  290  292  293  295  296
  297  298  299  301  302  303  304  305  306  307  308  309  310  312
  313  314  315  316  317  318  319  321  322  323  324  325  326  327
```

328	329	330	331	332	333	334	335	336	337	338	339	340	341
342	343	344	346	347	349	350	351	352	353	354	355	356	357
358	359	360	362	363	364	365	366	367	368	369	370	371	372
373	374	376	377	378	379	380	381	382	383	385	386	387	388
389	390	391	392	393	394	395	396	397	398	399	400	401	402
403	404	405	406	407	408	409	410	411	412	413	414	416	417
418	419	420	421	422	423	424	425	426	427	428	429	430	431
432	433	434	435	436	437	438	439	441	442	443	444	445	446
447	448	449	450	451	452	453	454	455	458	459	460	461	462
463	464	465	466	467	468	469	471	472	473	474	475	476	477
478	479	480	481	482	483	484	485	486	488	489	491	492	493
495	496	497	498	499	501	502	503	504	505	507	508	509	510
511	512	513	514	515	516	517	518	519	520	521	522	523	524
526	527	530	531	532	533	534	535	536	537	538	539	540	541
542	543	544	545	546	547	549	550	551	552	554	555	556	557
558	559	560	561	562	563	564	565	566	568	569	570	571	572
573	574	575	576	577	579	580	584	585	586	587	589	590	591
592	593	594	595	596	597	598	600	601	602	603	604	605	606
607	608	609	611	612	613	614	615	616	617	618	619	620	621
622	623	624	625	626	627	628	629	630	631	632	633	634	635
636	637	638	639	640	641	642	643	644	646	647	649	651	652
653	654	655	656	657	658	659	660	661	662	663	664	665	666
667	669	671	673	674	675	677	678	679	680	681	682	683	684
685	686	687	688	689	690	691	692	693	694	695	696	697	698
699	700	701	702	703	704	705	706	707	708	710	711	712	714
715	716	717	718	719	720	722	724	725	726	728	729	730	731
732	734	735	736	737	738	739	740	742	743	744	745	746	747
748	749	750	751	752	753	754	755	756	757	758	759	760	761
763	764	766	767	768	769	770	771	773	774	775	776	777	778
779	780	782	784	785	786	787	788	789	790	791	792	793	794
795	796	797	798	799	800	801	802	803	804	806	807	809	810
812	813	814	815	816	817	818	819	820	821	822	823	824	825
826	829	830	831	832	833	834	835	836	837	838	839	840	841
842	843	844	845	846	847	848	849	850	851	852	854	855	856
857	858	859	860	861	862	863	864	865	866	867	868	869	870
871	872	873	874	875	876	877	878	880	881	882	883	884	885
886	887	888	889	890	891	892	893	894	895	896	897	898	899
900	902	903	904	905	906	907	908	909	910	911	912	913	914
915	916	917	918	919	920	921	922	923	924	926	927	928	930
931	932	933	934	935	936	937	938	939	940	941	943	944	945
946	947	948	949	951	952	953	954	955	957	958	959	960	961
962	963	964	966	967	968	969	970	971	973	974	975	976	977
978	979	980	981	982	983	984	985	988	989	990	991	992	993
994	996	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009
1010	1011	1012	1013	1014	1015	1016	1018	1019	1020	1021	1022	1023	1026
1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1038	1039	1040	1042
1043	1044	1046	1047	1048	1049	1050	1052	1053	1054	1055	1056	1057	1058
1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1072	1073

```
 1074 1075 1076 1077 1078 1079]' is an invalid key

During handling of the above exception, another exception occurred:

InvalidIndexError                          Traceback (most recent call last)
<ipython-input-96-b2da531cf040> in <cell line: 13>()
     15     # Get training and test data
     16     train, test = I
---> 17     Xtr11 = X[train:]
     18     ytr11 = y[train]
     19     Xts11 = X[test,:]

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in␣
 ↪__getitem__(self, key)
   3777                 return self._getitem_multilevel(key)
   3778         # Do we have a slicer (on rows)?
-> 3779         indexer = convert_to_index_sliceable(self, key)
   3780         if indexer is not None:
   3781             if isinstance(indexer, np.ndarray):

/usr/local/lib/python3.10/dist-packages/pandas/core/indexing.py in␣
 ↪convert_to_index_sliceable(obj, key)
   2492     idx = obj.index
   2493     if isinstance(key, slice):
-> 2494         return idx._convert_slice_indexer(key, kind="getitem")
   2495
   2496     elif isinstance(key, str):

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪_convert_slice_indexer(self, key, kind)
   4282             indexer = key
   4283         else:
-> 4284             indexer = self.slice_indexer(start, stop, step)
   4285
   4286         return indexer

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪slice_indexer(self, start, end, step, kind)
   6557         self._deprecated_arg(kind, "kind", "slice_indexer")
   6558
-> 6559         start_slice, end_slice = self.slice_locs(start, end, step=step)
   6560
   6561         # return a slice

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪slice_locs(self, start, end, step, kind)
   6765         start_slice = None
   6766         if start is not None:
```

```
-> 6767                start_slice = self.get_slice_bound(start, "left")
   6768            if start_slice is None:
   6769                start_slice = 0

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪get_slice_bound(self, label, side, kind)
   6678            # we need to look up the label
   6679            try:
-> 6680                slc = self.get_loc(label)
   6681            except KeyError as err:
   6682                try:

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪get_loc(self, key, method, tolerance)
   3807                    #  InvalidIndexError. Otherwise we fall through and␣
 ↪re-raise
   3808                    #  the TypeError.
-> 3809                    self._check_indexing_error(key)
   3810                    raise
   3811

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪_check_indexing_error(self, key)
   5923                # if key is not a scalar, directly raise an error (the code
 ↪below
   5924                # would convert to numpy arrays and raise later any way) -␣
 ↪GH29926
-> 5925                raise InvalidIndexError(key)
   5926
   5927        @cache_readonly

InvalidIndexError: [   1    2    3    4    5    6    7    8    9   10   11   12 ␣
 ↪ 13   14
   15   16   17   18   20   21   22   23   24   25   26   27   28   29
   30   31   32   33   34   35   36   38   39   40   42   43   44   45
   46   47   48   49   50   51   52   53   54   55   56   59   60   62
   63   64   65   66   67   68   69   70   71   72   73   74   75   76
   77   78   79   80   81   82   84   85   86   87   89   90   91   92
   93   94   95   96   97   98   99  101  102  103  104  105  106  107
  108  109  110  111  112  113  114  115  117  119  121  122  123  124
  125  126  128  130  131  132  133  134  135  136  137  138  139  140
  141  142  143  144  145  146  148  149  150  151  152  153  154  155
  156  157  158  159  161  164  165  166  167  168  169  170  171  172
  173  174  175  176  177  178  179  180  181  182  183  184  185  186
  187  188  189  190  191  192  193  194  195  196  197  198  199  200
  201  202  203  204  205  206  207  208  209  210  211  213  215  216
  217  218  219  220  221  222  223  224  225  226  227  229  231  232
  233  234  235  236  237  238  239  240  241  242  243  245  246  247
```

248	249	250	251	252	253	254	255	256	257	258	259	260	261
263	264	265	266	269	270	272	273	274	275	276	277	278	279
280	281	282	283	284	285	286	288	289	290	292	293	295	296
297	298	299	301	302	303	304	305	306	307	308	309	310	312
313	314	315	316	317	318	319	321	322	323	324	325	326	327
328	329	330	331	332	333	334	335	336	337	338	339	340	341
342	343	344	346	347	349	350	351	352	353	354	355	356	357
358	359	360	362	363	364	365	366	367	368	369	370	371	372
373	374	376	377	378	379	380	381	382	383	385	386	387	388
389	390	391	392	393	394	395	396	397	398	399	400	401	402
403	404	405	406	407	408	409	410	411	412	413	414	416	417
418	419	420	421	422	423	424	425	426	427	428	429	430	431
432	433	434	435	436	437	438	439	441	442	443	444	445	446
447	448	449	450	451	452	453	454	455	458	459	460	461	462
463	464	465	466	467	468	469	471	472	473	474	475	476	477
478	479	480	481	482	483	484	485	486	488	489	491	492	493
495	496	497	498	499	501	502	503	504	505	507	508	509	510
511	512	513	514	515	516	517	518	519	520	521	522	523	524
526	527	530	531	532	533	534	535	536	537	538	539	540	541
542	543	544	545	546	547	549	550	551	552	554	555	556	557
558	559	560	561	562	563	564	565	566	568	569	570	571	572
573	574	575	576	577	579	580	584	585	586	587	589	590	591
592	593	594	595	596	597	598	600	601	602	603	604	605	606
607	608	609	611	612	613	614	615	616	617	618	619	620	621
622	623	624	625	626	627	628	629	630	631	632	633	634	635
636	637	638	639	640	641	642	643	644	646	647	649	651	652
653	654	655	656	657	658	659	660	661	662	663	664	665	666
667	669	671	673	674	675	677	678	679	680	681	682	683	684
685	686	687	688	689	690	691	692	693	694	695	696	697	698
699	700	701	702	703	704	705	706	707	708	710	711	712	714
715	716	717	718	719	720	722	724	725	726	728	729	730	731
732	734	735	736	737	738	739	740	742	743	744	745	746	747
748	749	750	751	752	753	754	755	756	757	758	759	760	761
763	764	766	767	768	769	770	771	773	774	775	776	777	778
779	780	782	784	785	786	787	788	789	790	791	792	793	794
795	796	797	798	799	800	801	802	803	804	806	807	809	810
812	813	814	815	816	817	818	819	820	821	822	823	824	825
826	829	830	831	832	833	834	835	836	837	838	839	840	841
842	843	844	845	846	847	848	849	850	851	852	854	855	856
857	858	859	860	861	862	863	864	865	866	867	868	869	870
871	872	873	874	875	876	877	878	880	881	882	883	884	885
886	887	888	889	890	891	892	893	894	895	896	897	898	899
900	902	903	904	905	906	907	908	909	910	911	912	913	914
915	916	917	918	919	920	921	922	923	924	926	927	928	930
931	932	933	934	935	936	937	938	939	940	941	943	944	945
946	947	948	949	951	952	953	954	955	957	958	959	960	961
962	963	964	966	967	968	969	970	971	973	974	975	976	977
978	979	980	981	982	983	984	985	988	989	990	991	992	993

```
  994  996  998  999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009
 1010 1011 1012 1013 1014 1015 1016 1018 1019 1020 1021 1022 1023 1026
 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1038 1039 1040 1042
 1043 1044 1046 1047 1048 1049 1050 1052 1053 1054 1055 1056 1057 1058
 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1072 1073
 1074 1075 1076 1077 1078 1079]
```

1.6 Multi-Class Classification

Now use the response variable in `df1['class']`. This has 8 possible classes. Use the `np.unique` funtion as before to convert this to a vector `y` with values 0 to 7.

[99]:
```python
# TODO 12

class_values = df1['class'].values
y, _ = np.unique(class_values, return_inverse=True)
```

Fit a multi-class logistic model by creating a `LogisticRegression` object, `logreg` and then calling the `logreg.fit` method.

Now perform 10-fold cross validation, and measure the confusion matrix `C` on the test data in each fold. You can use the `confustion_matrix` method in the `sklearn` package. Add the confusion matrix counts across all folds and then normalize the rows of the confusion matrix so that they sum to one. Thus, each element `C[i,j]` will represent the fraction of samples where `yhat==j` given `ytrue==i`. Print the confusion matrix. You can use the command

```
print(np.array_str(C, precision=4, suppress_small=True))
```

to create a nicely formatted print. Also print the overall mean and SE of the test accuracy across the folds.

[98]:
```python
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.metrics import confusion_matrix
predicted = cross_val_predict(logreg, X, y, cv=10)
confusion_matrices = [confusion_matrix(y, predicted) for y, predicted in zip(y,␣
 ↪predicted)]
C = sum(confusion_matrices)
C_normalized = C / C.sum(axis=1, keepdims=True)
print(np.array_str(C_normalized, precision=4, suppress_small=True))
accuracies = cross_val_score(logreg, X, y, cv=10)
mean_accuracy = accuracies.mean()
se_accuracy = accuracies.std() / np.sqrt(10)  # Assuming 10-fold␣
 ↪cross-validation

print(f"Mean Test Accuracy: {mean_accuracy:.4f}")
print(f"SE of Test Accuracy: {se_accuracy:.4f}")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
<ipython-input-98-63b71b10e10b> in <cell line: 4>()
      2 from sklearn.metrics import confusion_matrix
      3 predicted = cross_val_predict(logreg, X, y, cv=10)
----> 4 confusion_matrices = [confusion_matrix(y, predicted) for y, predicted i␣
 ↪zip(y, predicted)]
      5 C = sum(confusion_matrices)
      6 C_normalized = C / C.sum(axis=1, keepdims=True)

<ipython-input-98-63b71b10e10b> in <listcomp>(.0)
      2 from sklearn.metrics import confusion_matrix
      3 predicted = cross_val_predict(logreg, X, y, cv=10)
----> 4 confusion_matrices = [confusion_matrix(y, predicted) for y, predicted i␣
 ↪zip(y, predicted)]
      5 C = sum(confusion_matrices)
      6 C_normalized = C / C.sum(axis=1, keepdims=True)

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py in␣
 ↪confusion_matrix(y_true, y_pred, labels, sample_weight, normalize)
    315     (0, 2, 1, 1)
    316     """
--> 317     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    318     if y_type not in ("binary", "multiclass"):
    319         raise ValueError("%s is not supported" % y_type)

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py in␣
 ↪_check_targets(y_true, y_pred)
     84     y_pred : array or indicator matrix
     85     """
---> 86     check_consistent_length(y_true, y_pred)
     87     type_true = type_of_target(y_true, input_name="y_true")
     88     type_pred = type_of_target(y_pred, input_name="y_pred")

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in␣
 ↪check_consistent_length(*arrays)
    392     """
    393
--> 394     lengths = [_num_samples(X) for X in arrays if X is not None]
    395     uniques = np.unique(lengths)
    396     if len(uniques) > 1:

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in␣
 ↪<listcomp>(.0)
    392     """
    393
--> 394     lengths = [_num_samples(X) for X in arrays if X is not None]
    395     uniques = np.unique(lengths)
    396     if len(uniques) > 1:
```

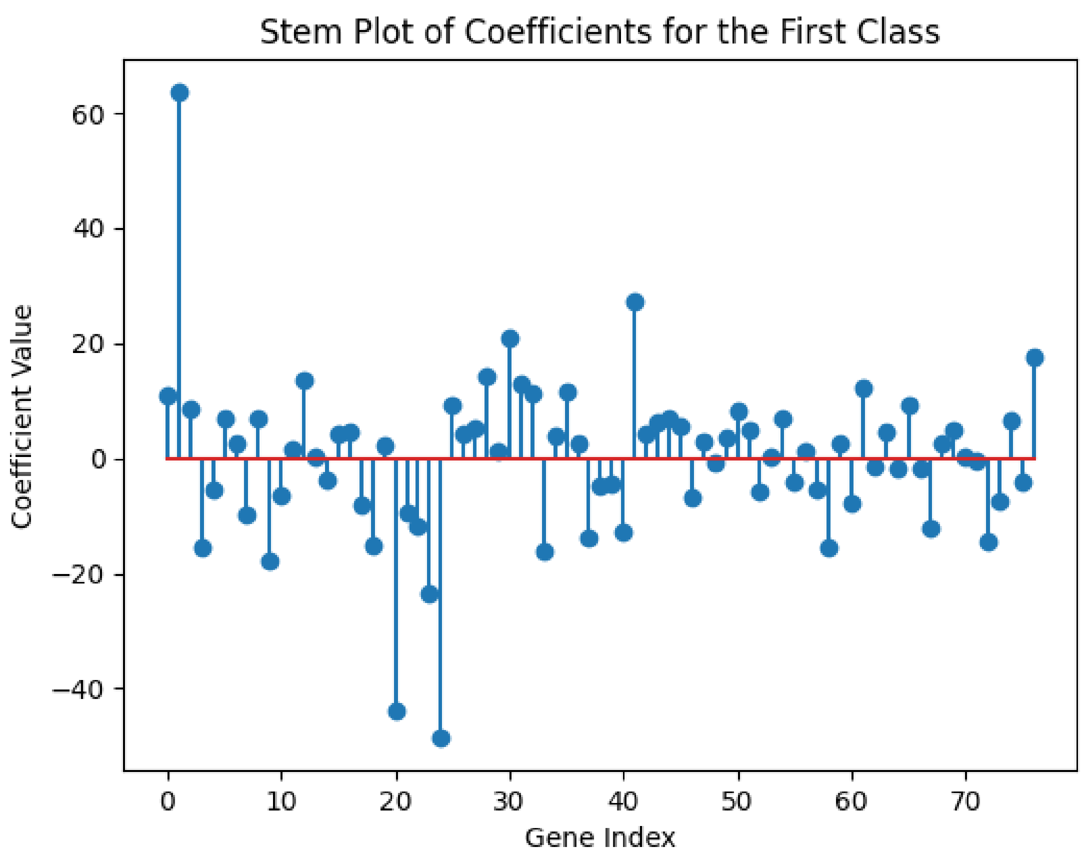
```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in␣
↪_num_samples(x)
    333     if hasattr(x, "shape") and x.shape is not None:
    334         if len(x.shape) == 0:
--> 335             raise TypeError(
    336                 "Singleton array %r cannot be considered a valid␣
↪collection." % x
    337             )

TypeError: Singleton array 'False' cannot be considered a valid collection.
```

Re-run the logistic regression on the entire training data and get the weight coefficients. This should be a 8 x 77 matrix. Create a stem plot of the first row of this matrix to see the coefficients on each of the genes.

[97]:
```
# TODO 14
W = logreg.coef_
first_class_coefficients = W[0]
indices = np.arange(len(first_class_coefficients))
plt.stem(indices, first_class_coefficients, use_line_collection=True)

plt.xlabel('Gene Index')
plt.ylabel('Coefficient Value')
plt.title('Stem Plot of Coefficients for the First Class')
```

```
<ipython-input-97-11f9b525da86>:5: MatplotlibDeprecationWarning: The
'use_line_collection' parameter of stem() was deprecated in Matplotlib 3.6 and
will be removed two minor releases later. If any parameter follows
'use_line_collection', they should be passed as keyword, not positionally.
  plt.stem(indices, first_class_coefficients, use_line_collection=True)
```

[97]:
```
Text(0.5, 1.0, 'Stem Plot of Coefficients for the First Class')
```



1.7 L1-Regularization

This section is bonus.

In most genetic problems, only a limited number of the tested genes are likely influence any particular attribute. Hence, we would expect that the weight coefficients in the logistic regression model should be sparse. That is, they should be zero on any gene that plays no role in the particular attribute of interest. Genetic analysis commonly imposes sparsity by adding an l1-penalty term. Read the `sklearn` documentation on the `LogisticRegression` class to see how to set the l1-penalty and the inverse regularization strength, `C`.

Using the model selection strategies from the housing demo, use K-fold cross validation to select an appropriate inverse regularization strength.
* Use 10-fold cross validation * You should select around 20 values of `C`. It is up to you find a good range. * Make appropriate plots and print out to display your results * How does the accuracy compare to the accuracy achieved without regularization.

```
[ ]: # TODO 15
```

```
[103]: import numpy as np
       from sklearn.linear_model import LogisticRegression
```

```python
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt


C_values = np.logspace(-5, 2, num=20)

mean_accuracies = []


nfold = 10
kf = KFold(n_splits=nfold, shuffle=True, random_state=1)

for C in C_values:
    accuracies = []
    for train, test in kf.split(X):
        X_train, X_test = X[train], X[test]
        y_train, y_test = y[train], y[test]


        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)


        logreg = LogisticRegression(penalty='l1', C=C, solver='liblinear')

        logreg.fit(X_train, y_train)


        y_pred = logreg.predict(X_test)

        accuracy = accuracy_score(y_test, y_pred)
        accuracies.append(accuracy)


    mean_accuracy = np.mean(accuracies)
    mean_accuracies.append(mean_accuracy)


best_C_index = np.argmax(mean_accuracies)
best_C = C_values[best_C_index]
plt.figure(figsize=(10, 6))
plt.semilogx(C_values, mean_accuracies, marker='o', linestyle='-', color='b')
plt.title('Accuracy vs. Inverse Regularization Strength (C)')
plt.xlabel('Inverse Regularization Strength (C)')
plt.ylabel('Mean Accuracy')
plt.grid(True)
plt.xticks(C_values, rotation=45)
plt.show()

print(f"Best C value: {best_C:.4e}")
print(f"Highest Mean Accuracy: {mean_accuracies[best_C_index]:.4f}")
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-103-04aed03dd304> in <cell line: 16>()
     17     accuracies = []
     18     for train, test in kf.split(X):
---> 19         X_train, X_test = X[train], X[test]
     20         y_train, y_test = y[train], y[test]
     21 

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in␣
 ↪__getitem__(self, key)
   3811             if is_iterator(key):
   3812                 key = list(key)
-> 3813             indexer = self.columns._get_indexer_strict(key, "columns")[ ]
   3814 
   3815         # take() does not accept boolean indexers

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪_get_indexer_strict(self, key, axis_name)
   6068             keyarr, indexer, new_indexer = self.
 ↪_reindex_non_unique(keyarr)
   6069 
-> 6070         self._raise_if_missing(keyarr, indexer, axis_name)
   6071 
   6072         keyarr = self.take(indexer)

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪_raise_if_missing(self, key, indexer, axis_name)
   6128                 if use_interval_msg:
   6129                     key = list(key)
-> 6130                 raise KeyError(f"None of [{key}] are in the␣
 ↪[{axis_name}]")
   6131 
   6132             not_found = list(ensure_index(key)[missing_mask.
 ↪nonzero()[0]].unique())

KeyError: "None of [Int64Index([   0,    1,    2,    3,    4,    5,    7,    8,  ␣
 ↪   9,   10,\n            ...\n           1070, 1071, 1072, 1073, 1074, 1075,␣
 ↪1076, 1077, 1078, 1079],\n          dtype='int64', length=972)] are in the␣
 ↪[columns]"
```