

Data Science Lab: Lab 4

Submit:

1. A pdf of your notebook with solutions.
2. A link to your colab notebook or also upload your .ipynb if not working on colab.

Goals of this Lab

1. Understanding Entropy
2. Scraping data.
3. Intro to Logistic Regression
4. Revisiting CIFAR-10 and MNIST.

✓ Daniel, Eric, Tony

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

Problem 1 (Optional)

Read Shannon's 1948 paper 'A Mathematical Theory of Communication'. Focus on pages 1-19 (up to Part II), the remaining part is more relevant for communication.

<https://math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf>

Problem 2: (Optional) Scraping, Entropy and ICML papers

ICML – the International Conference on Machine Learning – is a top research conference in Machine learning. Scrape all the pdfs of all ICML 2021 papers from <http://proceedings.mlr.press/v139/>, and answer the following questions.

Part 1. What are the top 10 common words in the ICML papers?

Part 2. Let Z be a randomly selected word in a randomly selected ICML paper. Estimate the entropy of Z .

Part 3. Synthesize a random paragraph using the marginal distribution over words.

Part 4. Synthesize a random paragraph using an n-gram model on words. Synthesize a random paragraph using any model you want. We will learn more about generating text in later classes when we get into NLP.

Note: downloading may take some time.

✓ Problem 3: Logistic Regression

The following is a logistic regression problem using a real data set, made available by the authors of the book "Applied Regression and Multilevel Modeling" by Gelman and Hill.

Download the data from the book, which you can find here <http://www.stat.columbia.edu/~gelman/arm/software/>. In particular, we are interested in the **arsenic** data set. The file **wells.dat** contains data on 3,020 households in Bangladesh. For each family, the natural arsenic level of each well was measured. In addition, the distance to the nearest safest well was measured. Each family is also described by a feature that relates to their community involvement, and a feature that gives the education level of the head of household. We are interested in building a model that predicts whether the family decided to switch wells or not, based on being informed of the level of arsenic in the well. Thus the "label" for this problem is the binary vector that is the first column of the dataset, labeled "switch."

Part 1. Fit a logistic regression model using only an offset term and the distance to the nearest safe well.

Part 2. Plot your answer: that is, plot the probability of switching wells as a function of the distance to the nearest safe well.

Part 3. Interpreting logistic regression coefficients: Use the "rule-of-4" discussed in class, to interpret the solution: what can you say about the change in the probability of switching wells, for every additional 100 meters of distance?

Part 4. Now solve a logistic regression incorporating the constant term, the distance and also arsenic levels. Report the coefficients


Part 5. Next we want to answer the question of which factor is more significant, distance, or arsenic levels? This is not a well specified question, since these two features have different units. One natural choice is to ask if after normalizing by the respective standard deviations of each feature, if moving one unit in one (normalized) feature predicts a larger change in probability of switching wells, than moving one unit in the other (also normalized) feature. Use this reasoning to answer the question.

Part 6. Now consider all the features in the data set. Also consider adding interaction terms among all features that have a large main effect. Use cross validation to build the best model you can (using your training set only), and then report the test error of your best model. (Note that since you have essentially unlimited access to your test set, this opens the door for massive overfitting. In contrast, Kaggle competitions try to mollify this by giving you only limited access to the test set.)

Part 7. (Optional) Now also play around with ℓ_1 and ℓ_2 regularization, and try to build the most accurate model you can (accuracy computed on the test data).


```
from google.colab import files
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```
uploaded = files.upload()
```

 Choose Files wells.dat

- wells.dat(n/a) - 107363 bytes, last modified: 9/27/2024 - 100% done

```
df_wells = pd.read_csv('wells.dat', sep='\s')
df_wells.head()
```

 <ipython-input-4-7d7a8ef61a13>:1: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex separ

```
df_wells = pd.read_csv('wells.dat', sep='\s')
```




	"switch"	"arsenic"	"dist"	"assoc"	"educ"
"1"	1	2.36	16.826000	0	0
"2"	1	0.71	47.321999	0	0
"3"	0	2.07	20.966999	0	10
"4"	1	1.15	21.486000	0	12
"5"	1	1.10	40.874001	1	14

Next steps: [View recommended plots](#) [New interactive sheet](#)

✓ Part 1

```
X = pd.DataFrame(df_wells.loc[:, "dist"])
y = df_wells.loc[:, "switch"]
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
model_logReg = LogisticRegression(penalty=None, fit_intercept=True)
model_logReg.fit(X_train, y_train)
```

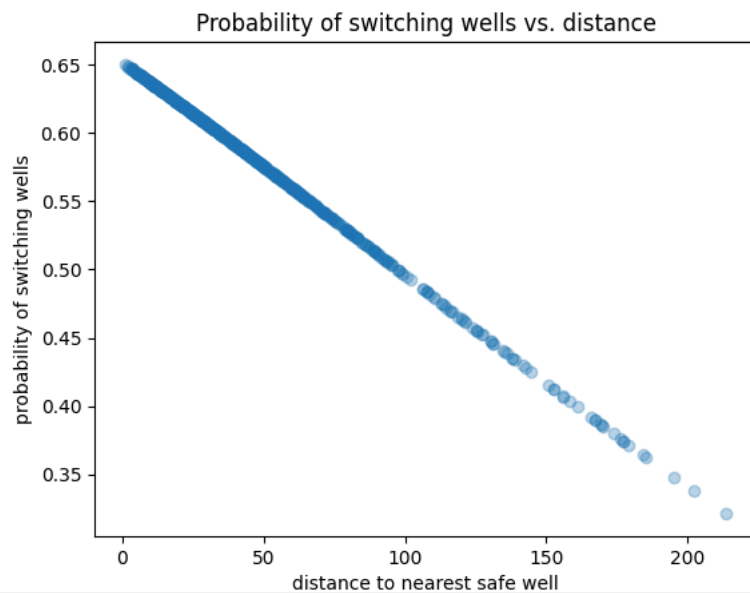
 **LogisticRegression**  

```
LogisticRegression(penalty=None)
```

✓ Part 2

```
pred = model_logReg.predict_proba(X_test)
prob_switch = pred[:, 1]
plt.scatter(X_test, prob_switch, alpha=0.3)
plt.xlabel('distance to nearest safe well')
plt.ylabel('probability of switching wells')
```

```
plt.title('Probability of switching wells vs. distance')
plt.show()
```



```
dist_array = X.values.tolist()
lins = model_logReg.coef_[0][0] * np.arange(len(dist_array)) + model_logReg.intercept_
probs = 1 / (1 + np.exp(-1 * lins))
print(model_logReg.coef_)

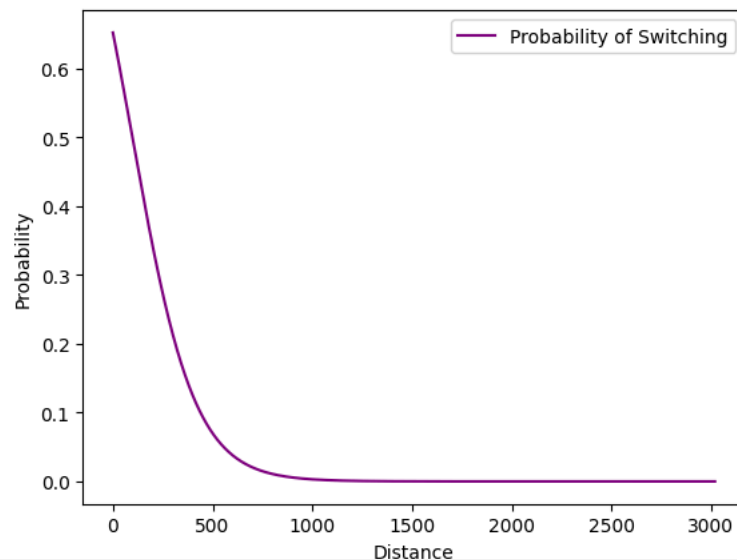
plt.plot(np.arange(len(dist_array)), probs, color='purple', label=f'Probability of Switching')

# Labels and title
plt.xlabel('Distance')
plt.ylabel('Probability')

# Add a legend
plt.legend()

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

```
[[[-0.00643066]]]
```



NOTE: The first plot looks linear, but this does not capture the entire story since the distances are from the test set. These distances are concentrated in a narrow range, flattening out the curve, making it look more linear.

The second plot is over a wide range of evenly spaced distance values. This range gives a clearer view of the full spectrum of switching probabilities as the distance changes, better reflecting the underlying logistic regression model's output. It makes sense that households will

not switch wells (probability approaches 0) as distances get very large.

▼ Part 3

```
ro4 = model_logReg.coef_/4 * 100
ro4[0][0]
```

```
↵ -0.17087270756143616
```

ANSWER:

This means that for every 100 meters, the probability of switching wells decreases by about 0.1709. This is the max change

▼ Part 4

```
X = pd.DataFrame(df_wells.loc[:, ['dist"', 'arsenic"']])
X_train, X_test, y_train, y_test = train_test_split(X, y)
model_logReg = LogisticRegression(penalty=None)
model_logReg.fit(X_train, y_train)
pred = model_logReg.predict(X_test)
print('Beta1 (distance):', model_logReg.coef_[0, 0])
print('Beta2 (arsenic levels):', model_logReg.coef_[0, 1])
```

```
↵ Beta1 (distance): -0.008444508235630073
   Beta2 (arsenic levels): 0.49572224960945843
```

ANSWER:

Beta1 (distance): -0.008444508235630073

Beta2 (arsenic levels): 0.49572224960945843

▼ Part 5

```
# normalize distance and arsenic
dist_mean = X.mean().iloc[0]
dist_std = X.std().iloc[0]

dist_norm = (X["dist"] - dist_mean) / dist_std

arsenic_mean = X.mean().iloc[1]
arsenic_std = X.std().iloc[1]
arsenic_norm = (X["arsenic"] - arsenic_mean) / arsenic_std

X_norm = pd.DataFrame({'dist': dist_norm, 'arsenic': arsenic_norm})
X_norm
```

```
↵
```

	dist	arsenic
"1"	-0.818788	0.634891
"2"	-0.026245	-0.855103
"3"	-0.711170	0.373013
"4"	-0.697682	-0.457772
"5"	-0.193818	-0.502923
...
"3016"	-0.753271	-1.026679
"3017"	-0.700280	-0.520984
"3018"	-1.055750	-1.035709
"3019"	-0.662441	-0.918315
"3020"	-0.714366	-0.900255

3020 rows x 2 columns

```
# train with normalized values
X_train, X_test, y_train, y_test = train_test_split(X_norm, y)
model_logReg = LogisticRegression(penalty=None)
model_logReg.fit(X_train, y_train)
pred = model_logReg.predict(X_test)
print('Beta1 (distance):', model_logReg.coef_[0, 0])
print('Beta2 (arsenic levels):', model_logReg.coef_[0, 1])
```

```
Beta1 (distance): -0.3396492453558449
Beta2 (arsenic levels): 0.5241022452282196
```

ANSWER:

Beta1 (distance): -0.3396492453558449

Beta2 (arsenic levels): 0.5241022452282196

When comparing the coefficients of normalized distance and arsenic levels, we can see that beta2 has a higher magnitude than beta1. This means that arsenic levels have a stronger impact than distance on whether wells will be switched. This makes sense since higher arsenic levels are more dangerous, so people would be more inclined to move than just based on distance.

Part 6

```
X_all = pd.DataFrame(df_wells.iloc[:, [1, 2, 3, 4]])
# interaction term between arsenic level and distance
arsenic_dist = X_all["arsenic"] * X_all["dist"]
X_all['arsenic x dist'] = arsenic_dist
X_all
```

	"arsenic"	"dist"	"assoc"	"educ"	arsenic x dist
"1"	2.36	16.826000	0	0	39.709361
"2"	0.71	47.321999	0	0	33.598619
"3"	2.07	20.966999	0	10	43.401688
"4"	1.15	21.486000	0	12	24.708900
"5"	1.10	40.874001	1	14	44.961401
...
"3016"	0.52	19.347000	1	5	10.060440
"3017"	1.08	21.386000	1	3	23.096880
"3018"	0.51	7.708000	0	4	3.931080
"3019"	0.64	22.841999	0	3	14.618879
"3020"	0.66	20.844000	1	5	13.757040

3020 rows x 5 columns

Next steps: [View recommended plots](#) [New interactive sheet](#)

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
```

```
X_train, X_test, y_train, y_test = train_test_split(X_all, y)
model_logReg = LogisticRegression(penalty=None)
cv_scores = cross_val_score(model_logReg, X_train, y_train, cv=5, scoring='accuracy')
```

```
print('Cross-validation scores:', cv_scores)
print('Mean cross-validation score:', np.mean(cv_scores))
```

```
model_logReg.fit(X_train, y_train)
pred = model_logReg.predict(X_test)
```

```
accuracy = accuracy_score(y_test, pred)
print('Test accuracy:', accuracy)
print('Test error:', 1-accuracy)
```

```
Cross-validation scores: [0.63355408 0.59381898 0.58498896 0.60485651 0.62030905]
Mean cross-validation score: 0.6075055187637969
```

Test accuracy: 0.6119205298013245
Test error: 0.3880794701986755

ANSWER:

Cross-validation scores: [0.63355408 0.59381898 0.58498896 0.60485651 0.62030905]

Mean cross-validation score: 0.6075055187637969

Test accuracy: 0.6119205298013245

Test error: 0.3880794701986755

Problem 4: Logistic Regression and CIFAR-10

In this problem you will explore the data set CIFAR-10, and you will use multinomial (multi-label) Logistic Regression to try to classify it. You will also explore visualizing the solution.

Part 1. (Optional) You can read about the CIFAR-10 and CIFAR-100 data sets here: <https://www.cs.toronto.edu/~kriz/cifar.html>.

Part 2. (Optional) OpenML curates a number of data sets. You will use a subset of CIFAR-10 provided by them. Read here for a description: <https://www.openml.org/d/40926>.

Part 3. Use the `fetch_openml` command from `sklearn.datasets` to import the CIFAR-10-Small data set. There are 20,000 data points. Do a train-test split on 3/4 - 1/4.

Part 4. Figure out how to display some of the images in this data set, and display a couple. While not high resolution, these should be recognizable if you are doing it correctly.


Part 5. You will run multi-class logistic regression on these using the cross entropy loss. You have to specify this specifically (`multi_class='multinomial'`). Use cross validation to see how good your accuracy can be. In this case, cross validate to find as good regularization coefficients as you can, for ℓ_1 and ℓ_2 regularization (i.e., for the Lasso and Ridge penalties), which are naturally supported in `sklearn.linear_model.LogisticRegression`. I recommend you use the solver `saga`. Note that this is quite a large problem: 20, 000 data points, each of them 3, 072-dimensional. Report your training and test losses.

Part 6. How sparse can you make your solutions without deteriorating your testing error too much? Here, I am asking you to try to obtain a sparse solution that has test accuracy that is close to the best solution you found.

Part 3

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
cifar_10 = fetch_openml('CIFAR-10_small')
```

```
X = cifar_10.data
X
```



	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	...	a3062	a3063	a3064	a3065	a3066	a3067	a3068	a3069	a3070	a3071
0	59	43	50	68	98	119	139	145	149	149	...	59	58	65	59	46	57	104	140	84	72
1	154	126	105	102	125	155	172	180	142	111	...	22	42	67	101	122	133	136	139	142	144
2	255	253	253	253	253	253	253	253	253	253	...	78	83	80	69	66	72	79	83	83	84
3	28	37	38	42	44	40	40	24	32	43	...	53	39	59	42	44	48	38	28	37	46
4	170	168	177	183	181	177	181	184	189	189	...	92	88	85	82	83	79	78	82	78	80
...
19995	76	76	77	76	75	76	76	76	76	78	...	228	185	177	223	239	239	235	236	234	236
19996	81	91	98	106	108	110	80	84	88	90	...	126	107	143	155	156	160	173	129	147	160
19997	20	19	15	15	14	13	12	11	10	9	...	114	112	68	50	52	52	51	50	53	47
19998	25	15	23	17	23	51	74	91	114	137	...	87	84	83	84	79	78	78	80	81	80
19999	73	98	99	77	59	146	214	176	125	218	...	84	89	88	85	93	93	90	94	58	26

20000 rows x 3072 columns

```
y = cifar_10.target
```

```
y
```

```
0      6
1      9
2      9
3      4
4      1
..
19995   8
19996   3
19997   5
19998   1
19999   7
Name: class, Length: 20000, dtype: category
Categories (10, object): ['0', '1', '2', '3', ..., '6', '7', '8', '9']
```

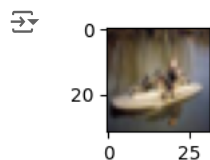
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, test_size=0.25)
```

Part 4

```
image_0 = X_train.iloc[0].values
```

```
image_0 = np.dstack((
    image_0[:1024].reshape(32, 32), # Red channel
    image_0[1024:2048].reshape(32, 32), # Green channel
    image_0[2048:].reshape(32, 32) # Blue channel
))
```

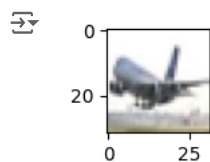
```
plt.figure(figsize=(1, 1))
plt.imshow(image_0)
plt.show()
```



```
image_1 = X_train.iloc[1].values
```

```
image_1 = np.dstack((
    image_1[:1024].reshape(32, 32), # Red channel
    image_1[1024:2048].reshape(32, 32), # Green channel
    image_1[2048:].reshape(32, 32) # Blue channel
))
```

```
plt.figure(figsize=(1, 1))
plt.imshow(image_1)
plt.show()
```



Part 5

```
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import log_loss, accuracy_score
```

```
Cs = np.logspace(-2, 1, 4)
Cs
```

```
array([ 0.01,  0.1 ,  1.  , 10.  ])
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
# train lasso model
model_logReg_l1 = LogisticRegressionCV(Cs=Cs, penalty='l1', multi_class='multinomial', solver='saga', max_iter=100, cv=5, scoring='neg_log_l1')
model_logReg_l1.fit(X_train, y_train)
```

```
LogisticRegressionCV
LogisticRegressionCV(Cs=array([ 0.01, 0.1 , 1. , 10. ]), cv=5,
multi_class='multinomial', n_jobs=-1, penalty='l1',
scoring='neg_log_loss', solver='saga', tol=0.01)
```

```
# train ridge model
model_logReg_l2 = LogisticRegressionCV(Cs=Cs, penalty='l2', multi_class='multinomial', solver='saga', max_iter=100, cv=5, scoring='neg_log_l1')
model_logReg_l2.fit(X_train, y_train)
```

```
LogisticRegressionCV
LogisticRegressionCV(Cs=array([ 0.01, 0.1 , 1. , 10. ]), cv=5,
multi_class='multinomial', n_jobs=-1,
scoring='neg_log_loss', solver='saga', tol=0.01)
```

```
l1_pred_train = model_logReg_l1.predict_proba(X_train)
cel_l1_train = log_loss(y_train, l1_pred_train)
print('L1 cross entropy training loss:', cel_l1_train)
```

```
l1_pred = model_logReg_l1.predict_proba(X_test)
cross_entropy_loss_l1 = log_loss(y_test, l1_pred)
print('L1 cross entropy testing loss:', cross_entropy_loss_l1)
```

```
pred = model_logReg_l1.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print('L1 test accuracy:', accuracy)
```

```
L1 cross entropy training loss: 1.4960455663056287
L1 cross entropy testing loss: 1.787272377619708
L1 test accuracy: 0.3798
```

```
l2_pred_train = model_logReg_l2.predict_proba(X_train)
cel_l2_train = log_loss(y_train, l2_pred_train)
print('L2 cross entropy training loss:', cel_l2_train)
```

```
l2_pred = model_logReg_l2.predict_proba(X_test)
cross_entropy_loss_l2 = log_loss(y_test, l2_pred)
print('L2 cross entropy testing loss:', cross_entropy_loss_l2)
```

```
pred = model_logReg_l2.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print('L2 test accuracy:', accuracy)
```

```
L2 cross entropy training loss: 1.474196292271934
L2 cross entropy testing loss: 1.8033390575434438
L2 test accuracy: 0.3738
```

ANSWER:

L1 cross entropy training loss: 1.4960455663056287

L1 cross entropy testing loss: 1.787272377619708

L1 test accuracy: 0.3798

L2 cross entropy training loss: 1.474196292271934

L2 cross entropy testing loss: 1.8033390575434438

L2 test accuracy: 0.3738

Part 6

```
# train lasso model for sparse solution
# lambda = 1/C so we want low C values for high lambda values
```



```
# high lambdas result in sparse solution
```

```
Cs = np.logspace(-4, 0, 5)
model_sparse = LogisticRegressionCV(Cs=C, penalty='l1', multi_class='multinomial', solver='saga', max_iter=100, cv=5, scoring='neg_log_loss')
model_sparse.fit(X_train, y_train)
```

```
LogisticRegressionCV(Cs=array([1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00]), cv=5,
multi_class='multinomial', n_jobs=-1, penalty='l1',
scoring='neg_log_loss', solver='saga', tol=0.01)
```

```
sparse_pred_train = model_sparse.predict_proba(X_train)
cel_sparse_train = log_loss(y_train, sparse_pred_train)
print('Sparse L1 cross entropy training loss:', cel_sparse_train)
```

```
sparse_pred = model_sparse.predict_proba(X_test)
cel_sparse = log_loss(y_test, sparse_pred)
print('Sparse L1 cross entropy testing loss:', cel_sparse)
```

```
pred = model_sparse.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print('Sparse L1 test accuracy:', accuracy)
```

```
Sparse L1 cross entropy training loss: 1.6268070293204913
Sparse L1 cross entropy testing loss: 1.7612106159016268
Sparse L1 test accuracy: 0.3902
```

```
# Aggregate cross-validation scores across all classes
mean_cv_scores = np.mean([np.mean(scores, axis=0) for scores in model_sparse.scores_.values()], axis=0)
print('Mean cross-validation scores for each penalty:', mean_cv_scores)
print('Penalties:', Cs)
```

```
Mean cross-validation scores for each penalty: [-1.96065098 -1.75092872 -1.75434318 -1.75752413 -1.75988494]
Penalties: [1.e-04 1.e-03 1.e-02 1.e-01 1.e+00]
```

ANSWER:

Sparse L1 cross entropy training loss: 1.6268070293204913

Sparse L1 cross entropy testing loss: 1.7612106159016268

Sparse L1 test accuracy: 0.3902

Mean cross-validation scores for each penalty: [-1.96065098 -1.75092872 -1.75434318 -1.75752413 -1.75988494]

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

We can see that a lambda of $1/1001 = 1000$, we achieve a similar accuracy and sparser solution. Note: the scores listed are negative log loss, so we consider the absolute value

✓ Problem 5: Multi-class Logistic Regression -- Visualizing the Solution

You will repeat the previous problem but for the MNIST data set which you will find here: <https://www.openml.org/d/554>. As we have seen before, MNIST is a data set of handwritten digits, and is considered one of the "easiest" image recognition problems in computer vision. We will see here how well logistic regression does, as you did above on the CIFAR-10 subset. In addition, we will see that we can visualize the solution, and that in connection to this, sparsity can be useful.

Part 1. Use the `fetch_openml` command from to import the MNIST data set, and choose a reasonable train-test split.

Part 2. Again run multi-class logistic regression on these using the cross entropy loss, as you did above. Try to optimize the hyperparameters. Report your training and test loss.

Part 3. Choose an ℓ_1 regularizer (penalty), and see if you can get a sparse solution with almost as good accuracy.

Part 4. Note that in Logistic Regression, the coefficients returned (i.e., the β 's) are the same dimension as the data. Therefore we can pretend that the coefficients of the solution are an image of the same dimension, and plot it. Do this for the 10 sets of coefficients that correspond to the 10 classes. You should observe that, at least for the sparse solutions, these "kind of" look like the digits they are classifying.

✓ Part 1

```

from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
mnist = fetch_openml('mnist_784')

X = mnist.data
y = mnist.target
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, test_size=0.25)

```

▼ Part 2

```

from sklearn.linear_model import LogisticRegressionCV

```

```

Cs = np.logspace(-2, 1, 4)
Cs

```

```

→ array([ 0.01,  0.1 ,  1.  , 10.  ])

```

```

# train lasso model
model_logReg_l1 = LogisticRegressionCV(Cs=Cs, penalty='l1', multi_class='multinomial', solver='saga', max_iter = 100, cv=5, scoring='neg_log
model_logReg_l1.fit(X_train, y_train)

```

```

→ LogisticRegressionCV
LogisticRegressionCV(Cs=array([ 0.01,  0.1 ,  1.  , 10.  ]), cv=5,
multi_class='multinomial', n_jobs=-1, penalty='l1',
scoring='neg_log_loss', solver='saga', tol=0.01)

```

```

# train ridge model
model_logReg_l2 = LogisticRegressionCV(Cs=Cs, penalty='l2', multi_class='multinomial', solver='saga', max_iter = 100, cv=5, scoring='neg_log
model_logReg_l2.fit(X_train, y_train)

```

```

→ LogisticRegressionCV
LogisticRegressionCV(Cs=array([ 0.01,  0.1 ,  1.  , 10.  ]), cv=5,
multi_class='multinomial', n_jobs=-1,
scoring='neg_log_loss', solver='saga', tol=0.01)

```

```

from sklearn.metrics import log_loss, accuracy_score

```

```

l1_pred_train = model_logReg_l1.predict_proba(X_train)
cel_l1_train = log_loss(y_train, l1_pred_train)
print('L1 cross entropy training loss:', cel_l1_train)

l1_pred = model_logReg_l1.predict_proba(X_test)
cross_entropy_loss_l1 = log_loss(y_test, l1_pred)
print('L1 cross entropy testing loss:', cross_entropy_loss_l1)

pred = model_logReg_l1.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print('L1 test accuracy:', accuracy)

→ L1 cross entropy training loss: 0.23052760546158166
L1 cross entropy testing loss: 0.2861509104721065
L1 test accuracy: 0.9235428571428571

```

```

l2_pred_train = model_logReg_l2.predict_proba(X_train)
cel_l2_train = log_loss(y_train, l2_pred_train)
print('L2 cross entropy training loss:', cel_l2_train)

l2_pred = model_logReg_l2.predict_proba(X_test)
cross_entropy_loss_l2 = log_loss(y_test, l2_pred)
print('L2 cross entropy testing loss:', cross_entropy_loss_l2)

pred = model_logReg_l2.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print('L2 test accuracy:', accuracy)

```

```

→ L2 cross entropy training loss: 0.22715840506759344
L2 cross entropy testing loss: 0.29062017293210163
L2 test accuracy: 0.9225714285714286

```

ANSWER:

L1 cross entropy training loss: 0.23052760546158166

L1 cross entropy testing loss: 0.2861509104721065

L1 test accuracy: 0.9235428571428571

L2 cross entropy training loss: 0.22715840506759344

L2 cross entropy testing loss: 0.29062017293210163

L2 test accuracy: 0.9225714285714286

▼ Part 3

```
# train lasso model for sparse solution
# lambda = 1/C so we want low C values for high lambda values
# high lambdas result in sparse solution
```

```
Cs = np.logspace(-4, 0, 5)
model_sparse = LogisticRegressionCV(Cs=C, penalty='l1', multi_class='multinomial', solver='saga', max_iter=100, cv=5, scoring='neg_log_loss')
model_sparse.fit(X_train, y_train)
```

↗ c:\Users\dnjnm\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\linear_model_logistic.py:1905: FutureWarning: 'multi_c' warnings.warn(

```
LogisticRegressionCV
LogisticRegressionCV(Cs=array([1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00]), cv=5,
multi_class='multinomial', n_jobs=-1, penalty='l1',
scoring='neg_log_loss', solver='saga', tol=0.01)
```

```
sparse_pred_train = model_sparse.predict_proba(X_train)
cel_sparse_train = log_loss(y_train, sparse_pred_train)
print('Sparse L1 cross entropy training loss:', cel_sparse_train)
```

```
sparse_pred = model_sparse.predict_proba(X_test)
cel_sparse = log_loss(y_test, sparse_pred)
print('Sparse L1 cross entropy testing loss:', cel_sparse)
```

```
pred = model_sparse.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print('Sparse L1 test accuracy:', accuracy)
```

↗ Sparse L1 cross entropy training loss: 0.23101294458636745
Sparse L1 cross entropy testing loss: 0.29642977675336213
Sparse L1 test accuracy: 0.9212

```
# Aggregate cross-validation scores across all classes
mean_cv_scores = np.mean([np.mean(scores, axis=0) for scores in model_sparse.scores_.values()], axis=0)
print('Mean cross-validation scores for each penalty:', mean_cv_scores)
print('Penalties:', Cs)
```

↗ Mean cross-validation scores for each penalty: [-0.38897093 -0.29316171 -0.29066496 -0.29172953 -0.29228033]
Penalties: [1.e-04 1.e-03 1.e-02 1.e-01 1.e+00]

ANSWER:

Sparse L1 cross entropy training loss: 0.23101294458636745

Sparse L1 cross entropy testing loss: 0.29642977675336213

Sparse L1 test accuracy: 0.9212

Mean cross-validation scores for each penalty: [-0.38897093 -0.29316171 -0.29066496 -0.29172953 -0.29228033]

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

We can see that a lambda of $1/1001 = 1000$, we achieve a similar accuracy and sparser solution. Note: the scores listed are negative log loss, so we consider the absolute value

Part 4

```
coefs = model_sparse.coef_  
coefs.shape
```

```
(10, 784)
```

```
fig, axs = plt.subplots(2, 5, figsize=(16, 7))
```

```
for i in range(10):  
    pixels = coefs[i, :].reshape(28, 28)  
    plot = plt.subplot(2, 5, i+1)  
    plot.set_title(f'Class {i}')  
    plot.imshow(pixels, cmap='Greys')
```

```
plt.show()
```

