

PROBLEM SET 2

You may also find the code here: <https://github.com/helenxtian/math-for-ml>.

4. a)

```
import numpy as np

##### Part a - main #####
# Load in training data and labels
# File available on Canvas

face_data_dict = np.load("face_emotion_data.npz")
face_features = face_data_dict["X"]
face_labels = face_data_dict["y"]
n, p = face_features.shape

# Solve the least-squares solution. weights is the
# array of weight coefficients
# TODO: find weights
XT_X = np.dot(face_features.T, face_features)
#  $X^T X$ 
XT_y = np.dot(face_features.T, face_labels)
#  $X^T y$ 
weights = np.dot(np.linalg.inv(XT_X), XT_y) #  $(X^T X)^{-1} X^T y$ 

print(f"Part 4a . Foundweights : \n{weights}")
```

```
helenxtian@Helens-MacBook-Air pset2_code % ...
Part 4a . Foundweights :
[[ 0.94366942]
 [ 0.21373778]
 [ 0.26641775]
 [-0.39221373]
 [-0.00538552]
 [-0.01764687]
 [-0.16632809]
 [-0.0822838 ]
 [-0.16644364]]
```

c)

```
##### Part b - function #####
def lstsq_cv_err(features: np.ndarray, labels: np.
ndarray, subset_count: int = 8) -> float:
    """Estimate the error of a least-squares
    classifier using cross-validation. Use
    subset_count different train/test splits with
    each subset acting as the holdout set once.
```

```

Parameters:
    features (np.ndarray): dataset features as a
        2D array with shape (sample_count ,
        feature_count)
    labels (np.ndarray): dataset class labels
        (+1/-1) as a 1D array with length (
        sample_count)
    subset_count (int): number of subsets to
        divide the dataset into
    Note: assumes that subset_count divides
        the dataset evenly

Returns:
    cls_err (float): estimated classification error
        rate of least-squares method"""

sample_count, feature_count = features.shape
subset_size = sample_count // subset_count

# Reshape arrays for easier subset-level
# manipulation
reshaped_feat = features.reshape(subset_count,
    subset_size, feature_count)
reshaped_lbls = labels.reshape(subset_count,
    subset_size)

subset_idcs = np.arange(subset_count)
train_set_size = (subset_count - 1) *
    subset_size
subset_err_counts = np.zeros(subset_count)

for i in range(subset_count):
    # TODO: select relevant dataset,
    # fit and evaluate a linear model,
    # then store errors in subset_err_counts[i]
    # Hint: you could extract the training
    # subset with train_subset_idcs =
    # subset_idcs[subset_idcs != i]

    test_feat = reshaped_feat[i]
    test_label = reshaped_lbls[i]

    train_subset_idcs = subset_idcs[subset_idcs
        != i]
    train_feat = reshaped_feat[train_subset_idcs
        ].reshape(train_set_size, feature_count)
    train_label = reshaped_lbls[
        train_subset_idcs].reshape(train_set_size
        )

```

```

XT_X = np.dot(train_feat.T, train_feat)
XT_y = np.dot(train_feat.T, train_label)
w = np.dot(np.linalg.inv(XT_X), XT_y)

y_pred = np.sign(np.dot(test_feat, w))
subset_err_counts[i] = np.sum(y_pred !=
    test_label)

# Average over the entire dataset to find the
    classification error
cls_err = np.sum(subset_err_counts) / (
    subset_count * subset_size)
return cls_err

```

```

##### Part b - main #####
# Run on the dataset with all features included
full_feat_cv_err = lstsq_cv_err(face_features ,
    face_labels)
print(f" P a r t 4b .      Errorestimate      :      {
    full_feat_cv_err*100:.3f}%")

```

```

helentian@Helens-MacBook-Air pset2_code % ...
P a r t 4b .      Errorestimate      : 4 .688%

```

e)

```

##### Part e - function #####
def drop_features_heuristic(features, labels,
    max_cv_err=0.06):
    chosen_features = list(range(features.shape[1]))
    best_err = lstsq_cv_err(features, labels)

    while len(chosen_features) > 1:
        reducedX = features[:, chosen_features]
        XT_X = reducedX.T @ reducedX
        XT_y = reducedX.T @ labels
        w = np.linalg.inv(XT_X) @ XT_y

        # absolute value weights, sorted from least
            to most
        min_idx = np.argmin(np.abs(w))
        feat_to_drop = chosen_features[min_idx]

        trial_features = [f for f in chosen_features
            if f != feat_to_drop]
        trialX = features[:, trial_features]
        trial_err = lstsq_cv_err(trialX, labels)

```

```

        if trial_err <= max_cv_err:
            chosen_features = trial_features
            best_err = trial_err
            print(f" P a r t 4e .   dropping   feature {
                feat_to_drop}, error: {trial_err
                *100:.2f}%")
        else:
            print(f" P a r t 4e .   dropping   feature {
                feat_to_drop}, error {trial_err
                *100:.2f}% > {max_cv_err*100:.2f}%")
            break

    return chosen_features, best_err

```

```

##### Part e - main #####
chosen_features, final_err =
    drop_features_heuristic(face_features,
        face_labels, max_cv_err=0.06)
print(" P a r t 4e .   Selectfeatures      :",
    chosen_features)
print(f" P a r t 4e .
        CErrorwithselectfeatures          :   {
        final_err*100:.2f}%")

```

```

helentian@Helens-MacBook-Air pset2_code % ...
P a r t 4e .   dropping   feature 4, error: 4.69%
P a r t 4e .   dropping   feature 5, error: 4.69%
P a r t 4e .   dropping   feature 7, error: 4.69%
P a r t 4e .   dropping   feature 8, error 6.25% > 6.00%
P a r t 4e .   Selectfeatures      : [0, 1, 2, 3, 6, 8]
P a r t 4e .   CErrorwithselectfeatures          :
        4   .69%

```

5.

```

import numpy as np
import matplotlib.pyplot as plt

# File available on Canvas
data = np.load('polydata_a24.npz')
x1 = np.ravel(data['x1'])
x2 = np.ravel(data['x2'])
y = data['y']

N = x1.size
p = np.zeros((3,N))

for d in [1,2,3]:

```

```

# Generate the X matrix for this d
# Note that here d is the degree of the polynomial,
# not the dimension of a vector
# Find the least-squares weight matrix w_d
# Evaluate the best-fit polynomial at each point (x1
# ,x2) # and store the result in the corresponding
# column of p
# Report the relative error of the polynomial fit

X = np.zeros((N, 2 * d + 1))
for i in range(N):
    X[i, 0] = 1.0
    for j in range(1, d + 1):
        X[i, j] = x1[i] ** j
        X[i, d + j] = x2[i] ** j

# (X^T X)^-1 X^T y
XT_X = np.dot(X.T, X)
XT_y = np.dot(X.T, y)
w_d = np.dot(np.linalg.inv(XT_X), XT_y)

y_hat = np.dot(X, w_d)
p[d - 1, :] = y_hat

rel_err = np.linalg.norm(y - y_hat) / np.linalg.norm(y)
print(f"d={d}:      relativeerror      =      {rel_err
      *100:.3f}%")

# Plot the degree 1 surface
Z1 = p[0,:].reshape(data['x1'].shape)
ax = plt.axes(projection='3d')
ax.scatter(x1,x2,y)
ax.plot_surface(data['x1'],data['x2'],Z1,color='orange')
plt.show()

# Plot the degree 2 surface
Z2 = p[1,:].reshape(data['x1'].shape)
ax = plt.axes(projection='3d')
ax.scatter(x1,x2,y)
ax.plot_surface(data['x1'],data['x2'],Z2,color='orange')
plt.show()

# Plot the degree 3 surface
Z3 = p[2,:].reshape(data['x1'].shape)
ax = plt.axes(projection='3d')
ax.scatter(x1,x2,y)
ax.plot_surface(data['x1'],data['x2'],Z3,color='orange')
plt.show()

```

```
helentian@Helens-MacBook-Air pset2_code % ...  
d=1:      relativeerror      = 32 .174%  
d=2:      relativeerror      = 18 .979%  
d=3:      relativeerror      = 2  .912%
```