Project 3: Poisonous Mushrooms

In this project, you'll investigate properties of mushrooms. This classic dataset contains over 8000 examples, where each describes a mushroom by a variety of features like color, odor, etc., and the target variable is an indicator for whether the mushroom is poisonous. The feature space has been binarized. Look at the feature_names below to see all 126 binary names.

You'll start by running PCA to reduce the dimensionality from 126 down to 2 so that you can easily visualize the data. In general, PCA is very useful for visualization (though sklearn.manifold.tsne is known to produce better visualizations). Recall that PCA is a linear transformation. The 1st projected dimension is the linear combination of all 126 original features that captures as much of the variance in the data as possible. The 2nd projected dimension is the linear combination of all 126 original features that captures as much of the remaining variance as possible. The idea of dense low dimensional representations is crucial to machine learning!

Once you've projected the data to 2 dimensions, you'll experiment with clustering using k-means and density estimation with Gaussian mixture models (GMM). Finally, you'll train a classifier by fitting a GMM for the positive class and a GMM for the negative class, and perform inference by comparing the probabilities output by each model.

As always, you're welcome to work on the project in groups and discuss ideas on the course wall, but please **prepare your own write-up and write your own code**.

```
%matplotlib inline

import urllib.request as urllib2 # For python3
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from matplotlib.colors import LogNorm
```

```
MUSHROOM_DATA = 'https://raw.githubusercontent.com/UCB-MIDS/207-Applied-Machine-Learn
MUSHROOM_MAP = 'https://raw.githubusercontent.com/UCB-MIDS/207-Applied-Machine-Learni
```

Load feature names.

```
feature_names = []
```

```
for line in urllib2.urlopen(MUSHROOM_MAP):
    [index, name, junk] = line.decode('utf-8').split()
    feature_names.append(name)

print('Loaded feature names: ', len(feature_names))
print(feature_names)

     Loaded feature names:  126
     ['cap-shape=bell', 'cap-shape=conical', 'cap-shape=convex', 'cap-shape=flat', 'c
```

Load data. The dataset is sparse, but there aren't too many features, so we'll use a dense representation, which is supported by all sklearn objects.

```
X, Y = [], []

for line in urllib2.urlopen(MUSHROOM_DATA):
    items = line.decode('utf-8').split()
    Y.append(int(items.pop(0)))
    x = np.zeros(len(feature_names))
    for item in items:
        feature = int(str(item).split(':')[0])
        x[feature] = 1
    X.append(x)

# Convert these lists to numpy arrays.
X = np.array(X)
Y = np.array(Y)

# Split into train and test data.
train_data, train_labels = X[:7000], Y[:7000]
test_data, test_labels = X[7000:], Y[7000:]

# Check that the shapes look right.
print(train_data.shape, test_data.shape)

     (7000, 126) (1124, 126)
```

Part 1:

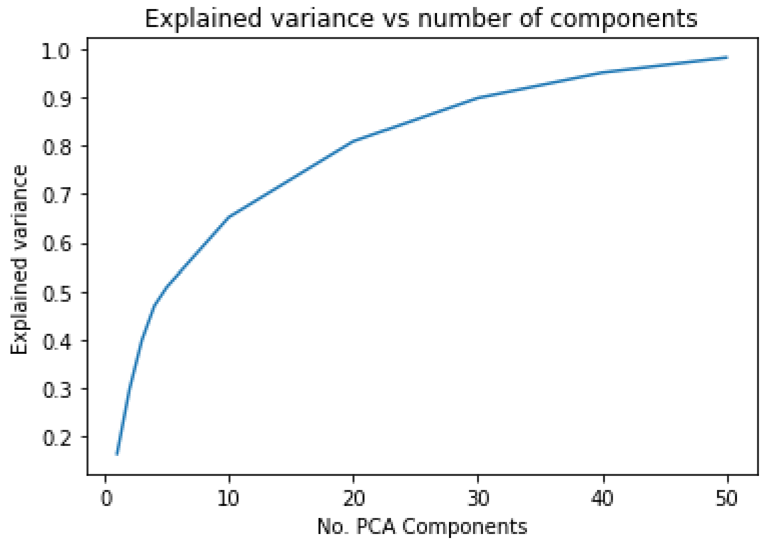
Do a principal components analysis on the data. Show what fraction of the total variance in the training data is explained by the first k principal components, for k in [1, 2, 3, 4, 5, 10, 20, 30, 40, 50]. Also show a lineplot of fraction of total variance vs. number of principal components, for all possible numbers of principal components.

Notes:

- You can use `PCA` to produce a PCA analysis.

```
def P1():
    ks = [1, 2, 3, 4, 5, 10, 20, 30, 40, 50]
    explained_variances = []

    for k in ks:
        pca = PCA(n_components=k)
        pca.fit(train_data)
        explained_variances.append(sum(pca.explained_variance_ratio_))
        print(f'k: {k}\t{sum(pca.explained_variance_ratio_)}')

    plt.title('Explained variance vs number of components')
    plt.xlabel('No. PCA Components')
    plt.ylabel('Explained variance')
    plt.plot(ks, explained_variances)


P1()
```

```
k: 1    0.164043312793342
k: 2    0.2972781014877562
k: 3    0.39901266670844077
k: 4    0.4696462310429905
k: 5    0.5083126898981211
k: 10   0.6522116573681314
k: 20   0.8088499373092883
k: 30   0.8984778373059245
k: 40   0.9509250682186428
k: 50   0.98210279348144
```



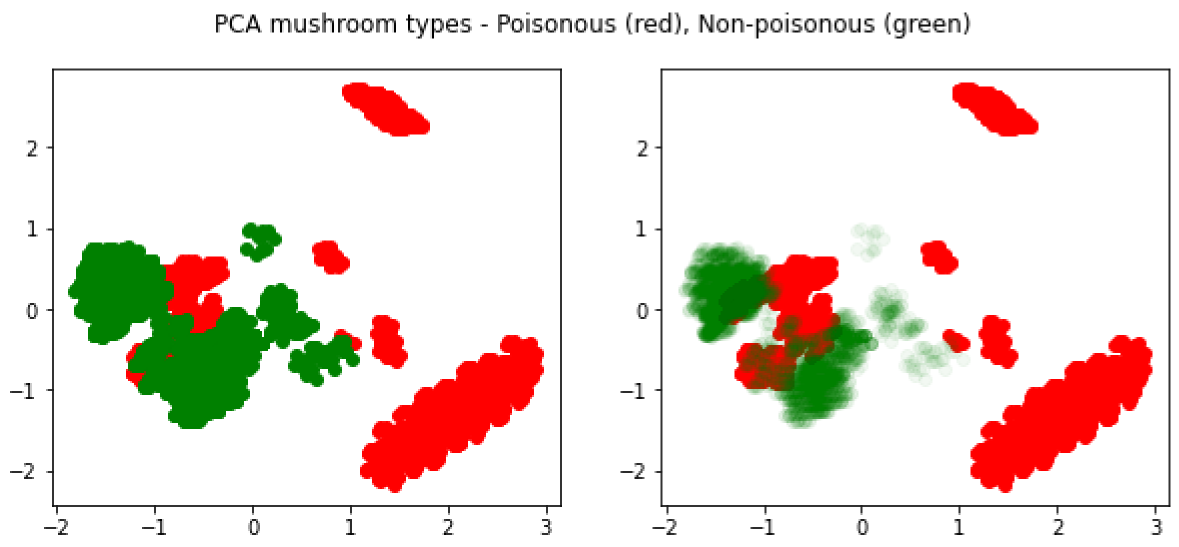
Part 2:

PCA can be very useful for visualizing data. Project the training data down to 2 dimensions and show as a square scatterplot. Show the positive (poisonous) examples in red and the negative

(non-poisonous) examples in green. Here's a reference for plotting:
http://matplotlib.org/users/pyplot_tutorial.html

Notes:

```
def P2():
    pca = PCA(n_components=2)
    reduced = pca.fit_transform(train_data)

    x = [r[0] for r in reduced]
    y = [r[1] for r in reduced]

    poison_x, poison_y = [], []
    non_ps_x, non_ps_y = [], []

    for x_, y_, label in zip(x, y, train_labels):
        if label == 1:
            poison_x.append(x_)
            poison_y.append(y_)
        else:
            non_ps_x.append(x_)
            non_ps_y.append(y_)

    fig, axs = plt.subplots(ncols=2, figsize=(10, 4))

    fig.suptitle('PCA mushroom types - Poisonous (red), Non-poisonous (green)')
    axs[1].plot(poison_x, poison_y, 'ro', alpha=1)
    axs[1].plot(non_ps_x, non_ps_y, 'go', alpha=0.05)
    axs[0].plot(poison_x, poison_y, 'ro', alpha=1)
    axs[0].plot(non_ps_x, non_ps_y, 'go', alpha=1)

P2()
```



Part 3:

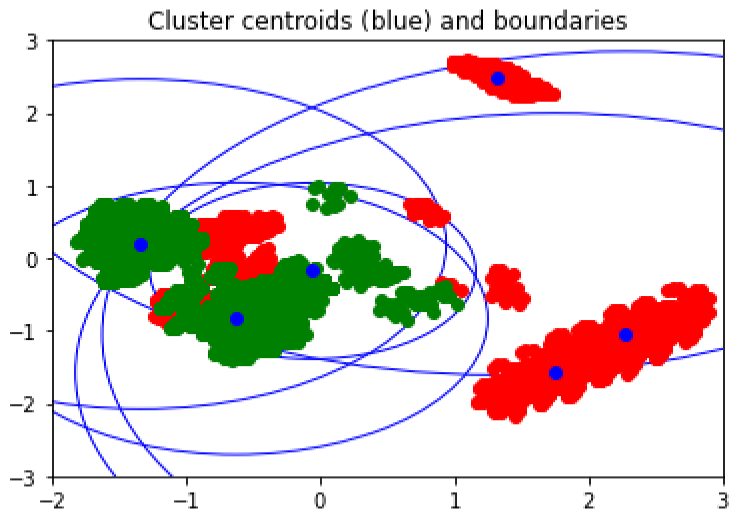
Fit a k-means cluster model with 6 clusters over the 2d projected data. As in part 2, show as a square scatterplot with the positive (poisonous) examples in red and the negative (non-poisonous) examples in green. For each cluster, mark the centroid and plot a circle that goes through the cluster's example that is most distant from the centroid.

Notes:

- You can use `KMeans` to produce a k-means cluster analysis.
- You can use `linalg.norm` to determine distance (dissimilarity) between observations.

```
def P3():
    pca = PCA(n_components=2)
    reduced_train_data = pca.fit_transform(train_data)
    poison_x, poison_y = [], []
    non_ps_x, non_ps_y = [], []

    x = [r[0] for r in reduced_train_data]
    y = [r[1] for r in reduced_train_data]

    for x_, y_, label in zip(x, y, train_labels):
        if label == 1:
            poison_x.append(x_)
            poison_y.append(y_)
        else:
            non_ps_x.append(x_)
            non_ps_y.append(y_)

    km = KMeans(n_clusters=6)
    km.fit(reduced_train_data)

    plt.plot(poison_x, poison_y, 'ro', alpha=1)
    plt.plot(non_ps_x, non_ps_y, 'go', alpha=1)

    distances =  [0] * 6
    for x, y in reduced_train_data:
        cluster = km.predict([[x, y]])[0]
        cx, cy = km.cluster_centers_[cluster]
        d = np.linalg.norm([[x, y], [cx, cy]])
        distances[cluster] = max(d, distances[cluster])

    ax = plt.axes()
    extent = ax.get_window_extent()
    for (x, y), r in zip(km.cluster_centers_, distances):
        plt.plot([x], [y], 'bo')
        ax.add_patch(plt.Circle([x, y], r, facecolor='#00000000', edgecolor='blue'))
    ax.axis([-2, 3.0, -3, 3])
    plt.title('Cluster centroids (blue) and boundaries')

P3()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:31: MatplotlibDepre
```



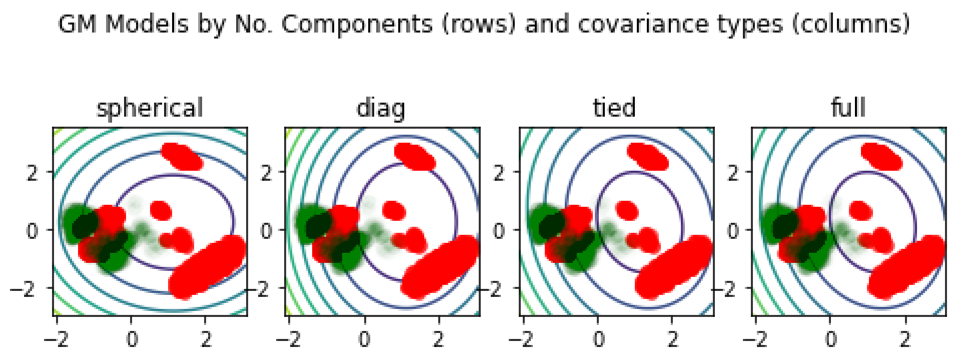
Part 4:

Fit Gaussian mixture models for the positive (poisonous) examples in your 2d projected data. Vary the number of mixture components from 1 to 4 and the covariance matrix type 'spherical', 'diag', 'tied', 'full' (that's 16 models). Show square plots of the estimated density contours presented in a 4x4 grid - one row each for a number of mixture components and one column each for a convariance matrix type.

Notes:

- You can use `GaussianMixture(n_components=..., covariance_type=..., random_state=12345)` to produce a Gaussian mixture model.
- You can use `contour` in combination with other methods to plot contours, like in this example: http://scikit-learn.org/stable/auto_examples/mixture/plot_gmm_pdf.html#example-mixture-plot-gmm-pdf-py
- You can use `contour` without the `norm` and `levels` parameters.

```
def P4():
    import random
    pca = PCA(n_components=2)
    reduced_train_data = pca.fit_transform(train_data)
    poison_x, poison_y = [], []
    non_ps_x, non_ps_y = [], []

    x = [r[0] for r in reduced_train_data]
    y = [r[1] for r in reduced_train_data]

    for x_, y_, label in zip(x, y, train_labels):
        if label == 1:
            poison_x.append(x_)
            poison_y.append(y_)
        else:
            non_ps_x.append(x_)
            non_ps_y.append(y_)

    fig, axs = plt.subplots(4, 4, figsize=(8, 8))
    fig.suptitle('GM Models by No. Components (rows) and covariance types (columns)')
    for i, matrix_type in enumerate(['spherical', 'diag', 'tied', 'full']):
        for n in range(1, 5):
            gm = GaussianMixture(n_components=n, covariance_type=matrix_type, random_
            gm.fit(list(zip(poison_x, poison_y)))

            ax = axs[n - 1, i]
            if n == 1:
                ax.set_title(matrix_type)

            ax.plot(poison_x, poison_y, 'ro', alpha=0.5)
            ax.plot(non_ps_x, non_ps_y, 'go', alpha=0.01)

            x = np.linspace(-2.1, 3.1)
            y = np.linspace(-3., 3.5)
            X, Y = np.meshgrid(x, y)
            XX = np.array([X.ravel(), Y.ravel()]).T
            Z = -gm.score_samples(XX)
            Z = Z.reshape(X.shape)

            ax.contour(X, Y, Z)

    plt.show()

P4()
```



Part 5:

Fit two Gaussian mixture models, one for the positive examples and one for the negative examples in your 2d projected data. Use 4 mixture components and full convariance for each model. Predict the test example labels by picking the labels corresponding to the larger of the two models' probabilities. What is the accuracy of you predictions on the test data?

Notes:

- You can use `GaussianMixture(n_components=..., covariance_type=..., random_state=12345)` to produce a Gaussian mixture model.
- You can use `GaussianMixture`'s `score_samples` method to find the probabilities.

```
from sklearn.metrics import accuracy_score

def P5():
    # Create our model with 2 components (2-dimensions) and extract the data for our
    pca_model = PCA(n_components=2)
    pca_data = pca_model.fit_transform(train_data)
    pca_test = pca_model.transform(test_data)

    # Create our first GMM fitted only for the positive (poisonous) examples
    gmm_positive = GaussianMixture(n_components=4, covariance_type='full', random_sta
    gmm_positive.fit(pca_data[train_labels==1])
    gmm_positive_results = gmm_positive.predict(pca_test)
    positive_prob = gmm_positive.score_samples(pca_test)
    print("Accuracy of positive model = {}".format(accuracy_score(test_labels, gmm_po

    # Create a second GMM fitted only for the negative (non-poisonous) examples
    gmm_negative = GaussianMixture(n_components=4, covariance_type='full', random_sta
    gmm_negative.fit(pca_data[train_labels==0])
    gmm_negative_results = gmm_negative.predict(pca_test)
    negative_prob = gmm_negative.score_samples(pca_test)
    print("Accuracy of negative model = {}".format(accuracy_score(test_labels, gmm_ne

    # This list will hold the label of the greater of the two probabilities from the
    greatest_prob = []
    # For each element in the positive and negative probabilities list, pick the grea
    for i in range(0, len(negative_prob)):
        if positive_prob[i] >= negative_prob[i]:
            greatest_prob.append(1)
        else:
            greatest_prob.append(0)

    print(greatest_prob)
    print(test_labels)
    print("Accuracy of greatest probabilites model = {}".format(accuracy_score(test_l

    #
    # Start of my code
    #
    # It looks like this problem was already solved?
    # And I looked at the mtime: Jul 21, 2021 by Clarence Chio Wen Han
    # So it was modified before Jul 26

P5()
```

```
Accuracy of positive model = 0.2099644128113879
Accuracy of negative model = 0.19572953736654805
[0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
[0 0 0 ... 1 0 1]
Accuracy of greatest probabilites model = 0.9501779359430605
```

Part 6:

Run a series of experiments to find the Gaussian mixture model that results in the best accuracy with no more than 50 parameters. Do this by varying the number of PCA components, the number of GMM components, and the covariance type.

Notes:

- You can use `GaussianMixture(n_components=..., covariance_type=..., random_state=12345)` to produce a Gaussian mixture model.
- For spherical, diag, and full covariance types:
  - number of parameters = (number of parameters per gmm component * number of gmm components - 1) * number of classes
  - number of parameters per gmm component includes all the means plus all the non-zero, non-duplicated values in the covariance matrix plus the mixing weight
  - Each mixing weight parameter indicates how much to weight a particular gmm component; the -1 above accounts for the fact that the mixing weights must sum to 1, so you do not need to include the last mixing weight as its own parameter
- To calculate the number of parameters for tied covariance type:
  - number of parameters = (number of parameters per class - 1) * number of classes

  - number of parameters per class includes all the means and mixing weights for all the gmm components plus all the non-zero, non-duplicated values in the one shared covariance matrix
  - Each mixing weight parameter indicates how much to weight a particular gmm component; the -1 above accounts for the fact that the mixing weights must sum to 1, so you do not need to include the last mixing weight as its own parameter

```
def P6():
    n_classes = 2

    best_accuracy = 0
    for n_vars in range(1, 50):
        for n_components in range(1, 50):
            for covariance_type in ['spherical', 'diag', 'tied', 'full']:
                if covariance_type == 'spherical':
                    n_params_per_gaussian = n_vars + 1
                    n_params = n_params_per_gaussian * n_components * n_classes
                if covariance_type == 'diag':
                    n_params_per_gaussian = n_vars + n_vars
                    n_params = n_params_per_gaussian * n_components * n_classes
                if covariance_type == 'tied':
                    n_params_per_gaussian = (n_vars * n_components) + (n_vars + (n_va
                    n_params = n_params_per_gaussian * n_classes
                if covariance_type == 'full':
                    n_params_per_gaussian = n_vars + (n_vars + (n_vars * n_vars - n_v
                    n_params = n_params_per_gaussian * n_components * n_classes


            if n_params <= 50:
                pca = PCA(n_components=n_vars)
                pca_data = pca.fit_transform(train_data)
                pca_test = pca.transform(test_data)

                gmm_p = GaussianMixture(n_components=n_components, covariance_type=co
                gmm_n = GaussianMixture(n_components=n_components, covariance_type=co
                positives = pca_data[train_labels == 1]
                negatives = pca_data[train_labels != 1]

                gmm_p.fit(positives)
                gmm_n.fit(negatives)

                scores_p = gmm_p.score_samples(pca_test)
                scores_n = gmm_n.score_samples(pca_test)
                score_labels = [1 if p > n else 0 for p, n in zip(scores_p, scores_n)
                accuracy = accuracy_score(test_labels, score_labels)
                result = (n_vars, n_components, covariance_type, n_params)

                if accuracy > best_accuracy:
                    best_accuracy = accuracy
                    best_result = result

    best_n_vars, best_n_components, best_covariance_type, best_n_params = best_result
    print(f'The best GMM had {best_n_vars} PCA components, {best_n_components} GMM co
    print(f'It had an accuracy of {best_accuracy}')

P6()
```

```
        The best GMM had 2 PCA components, 5 GMM components, full covariance type, and 5
        It had an accuracy of 0.9564056939501779
```

check 2s completed at 1:07 PM