ECE 57000 Assignment 5 Exercise

Your Name: Mohmmad Alwakeel

For this assignment, you will implement and explore various density estimation methods.

Exercise 1: Density estimation in 1D

In this exercise, you will write code to estimate 1D densities. Specifically, you will write code to estimate a Gaussian density, a histogram density, and a kernel density.

Task 1.1: Gaussian density

For this first one you will estimate a Gaussian density via MLE. As discussed in class, this simplifies to estimating the mean and standard devaition of the data and using these empirical estimates for the Gaussian distribution. The Gaussian PDF can be evaluated using the function `scipy.stats.norm.pdf` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html).

In [377]:
```
import numpy as np
import scipy.stats
from sklearn.base import BaseEstimator
class GaussianDensity(BaseEstimator):
    def fit(self, X, y=None):
        ##### Your code here #####
        # You should estimate the mean and variance of the data  and save as s
elf.mean_ and self.std_
        # (note that X will be shape (n,1) because there is only 1 feature).

        self.mean_ = X.mean()
        self.std_ = X.std()

        ############################
        return self

    def predict_proba(self, X):
        ##### Your code here #####
        # This should return the PDF values for each sample in X (again of sha
pe (n, 1))
        # This should use your self.mean_ and self.std_ variables saved from t
he fit method

        pdf_values = scipy.stats.norm.pdf(X, loc=self.mean_, scale=self.std_).
ravel()
            #value = (self.std_**2) / (self.mean_ - sample)


        return pdf_values  # Output should be of shape (n,), i.e., a 1D array
        ############################
```

Task 1.2: Histogram density

Now you will implement a histogram density estimate given min, max and number of bins. The function `np.searchsorted` (https://numpy.org/doc/stable/reference/generated/numpy.searchsorted.html) may be useful but is not required.

NOTE: The value of the histogram outside of the min max values should be set to 0.

In [414]:
```
import numpy as np
import scipy.stats
from sklearn.base import BaseEstimator
class HistogramDensity(BaseEstimator):
    def __init__(self, n_bins, min_val, max_val):
        self.n_bins = n_bins
        self.min_val = min_val
        self.max_val = max_val

    def fit(self, X, y=None):
        ##### Your code here #####
        # First create equally spaced bin_edges based on min_val, max_val and
 n_bins
        #  and save as self.bin_edges_
        #  (note the shape of self.bin_edges_ should be (n_bins+1,) )
        # Second, estimate the frequency for each bin based on the input data
 X
        #  (i.e., the number of training samples that fall into that bin divid
ed
        #  by the total number of samples)
        # Third, using the probability for each bin, compute the density value
 (i.e., PDF) for
        #  each bin. (Note you will have to account for the width of the bin t
o ensure
        #  that integrating your density function from min_value to max_value
 will be 1).
        #  Save the density per bin as self.pdf_per_bin_ which should have the
 shape (n_bins,)


        self.bin_edges_ = np.linspace(self.min_val, self.max_val, self.n_bins
+ 1)
        freq = np.zeros(self.bin_edges_.shape[0])

        #freq = np.searchsorted(self.bin_edges_,X)
        for x in X:

            val = np.searchsorted(self.bin_edges_,x,side='left')

            freq[val-1] += 1

        #freq = np.divide(freq,len(X))
        freq = np.array(freq/len(X))

        #self.pdf_per_bin_ = []
        width = self.bin_edges_[len(self.bin_edges_) - 1] - self.bin_edges_[le
n(self.bin_edges_) - 2]

        self.pdf_per_bin_ = np.array(freq/width)


        ############################
        return self.pdf_per_bin_

    def predict_proba(self, X):
        ##### Your code here #####
```

```
        # You should return the PDF value of the samples X.  This requires fin
ding out which
        #  bin each sample falls into and returning it's corresponding density
value
        #  **Importantly, if the value is less than min_value or greater than
 max_value,
        #    then a pdf value of 0 should be returned.



        bin_edges_ = list(self.bin_edges_)


        pdf_values = []
        for x in X:
            if x < self.bin_edges_[0] or x >= self.bin_edges_[len(self.bin_edg
es_)-1]:
                pdf_values.append(0)

            else:
                for value in self.bin_edges_:
                    if x <= value:

                        val = np.searchsorted(self.bin_edges_,x)
                        pdf_values.append(self.pdf_per_bin_[val-1])

                        break

        pdf_values = np.array(pdf_values)

        return pdf_values  # Output should be of shape (n,), i.e., a 1D array
        ###########################
```

In [ ]:

Task 1.3: Kernel density

Now you will implement a kernel density estimate (KDE) via a Gaussian kernel given the bandwidth parameter (i.e., the standard deviation of the Gaussian kernel. Specifically, the Gaussian kernel density is given by:

$$p(x;\mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} p_{\mathcal{N}}(x;\mu = x_i, \sigma = h)$$

where $\mathcal{D} = \{x_i\}_{i=1}^{n}$ is a training dataset of n samples, $p_{\mathcal{N}}$ is the Gaussian/normal density function and h is called the bandwidth hyperparameter of the KDE model. (Note that fitting merely requires saving the training dataset and using it to compute densities of new samples.)

In [372]:
```
import numpy as np
import scipy.stats
from sklearn.base import BaseEstimator
class KernelDensity(BaseEstimator):
    def __init__(self, bandwidth):
        self.bandwidth = bandwidth

    def fit(self, X, y=None):
        ##### Your code here #####
        # Save the training data in self.X_train_
        self.X_train_ = X
        ############################
        return self.X_train_

    def predict_proba(self, X):
        ##### Your code here #####
        # You should return the KDE PDF value of the samples X.
        #  Note that the sum above is over the TRAINING samples, not the test
 samples
        #  so you should use the samples saved in the fit method.
        #scipy.stats.gaussian_kde(X,)
        pdf_value = 0
        for x in self.X_train_:
            pdf_value += scipy.stats.norm.pdf(X,x,scale= self.bandwidth)

        pdf_values = (pdf_value/len(self.X_train_)).ravel()


        return pdf_values  # Output should be of shape (n,), i.e., a 1D array
        ############################
```

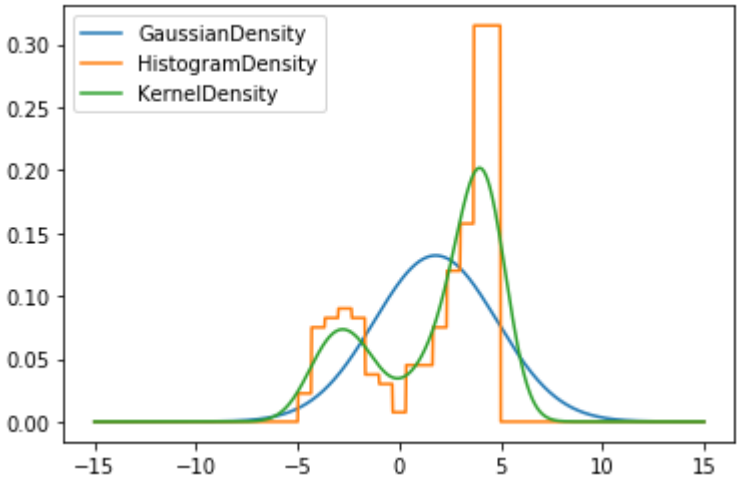
Test code

The following code will test each of your density estimators. Run this to test your code and this is what will be used for grading your code.

In [541]:
```
import scipy.stats
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
# Generate some data and split into train and test
min_val, max_val = -5, 5
diff = max_val - min_val
X = diff * np.vstack([scipy.stats.beta(6,1).rvs(size=(300,1)), scipy.stats.bet
a(2,7).rvs(size=(100,1))]) - diff/2
X_train, X_test = train_test_split(X, test_size=0.5, random_state=15)
print(X_train.shape, X_test.shape)

# Loop through models
models = [GaussianDensity(),
          HistogramDensity(n_bins=15, min_val=min_val, max_val=max_val),
          KernelDensity(bandwidth=1)
         ]

for model in models:
    print(f'Fitting {type(model).__name__} model')
    # Fit models
    model.fit(X_train)

    # Sanity checks
    xq = np.linspace(min_val-diff, max_val+diff, num=1000)
    pdf_vals = model.predict_proba(xq.reshape(-1, 1))
    # Check that right size and >= 0
    print(f'{len(pdf_vals.shape) == 1 and pdf_vals.shape[0] == len(xq)}, Shape
={pdf_vals.shape}'
          f' - Is the output the correct shape?')
    print(f'{np.all(pdf_vals>=0)}, Num neg={np.sum(pdf_vals < 0)} - Are all pd
f values >= 0? ')

    # Check that integrates to 1 vai approximate numerical integration
    model_pdf = lambda x: model.predict_proba(np.array(x).reshape(1,1))[0]
    quad_out = scipy.integrate.quad(model_pdf, min_val - diff, max_val + diff,
limit=100, full_output=True)
    print(f'{np.abs(quad_out[0] - 1) < 1e-4}, quad_out={quad_out[0]} - Does th
e PDF integrate to 1? ')
    print('')

    # Plot density model
    plt.plot(xq, pdf_vals, label=type(model).__name__)

plt.legend()
```

```
(200, 1) (200, 1)
Fitting GaussianDensity model
True, Shape=(1000,) - Is the output the correct shape?
True, Num neg=0 - Are all pdf values >= 0?
True, quad_out=0.9999939051919124 - Does the PDF integrate to 1?

Fitting HistogramDensity model
True, Shape=(1000,) - Is the output the correct shape?
True, Num neg=0 - Are all pdf values >= 0?
True, quad_out=0.99996454271539 - Does the PDF integrate to 1?

Fitting KernelDensity model
True, Shape=(1000,) - Is the output the correct shape?
True, Num neg=0 - Are all pdf values >= 0?

D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:72: VisibleDeprecationWa
rning: Creating an ndarray from ragged nested sequences (which is a list-or-t
uple of lists-or-tuples-or ndarrays with different lengths or shapes) is depr
ecated. If you meant to do this, you must specify 'dtype=object' when creatin
g the ndarray

True, quad_out=1.0 - Does the PDF integrate to 1?
```

Out[541]: <matplotlib.legend.Legend at 0x267cd67df08>



In [ ]:

Exercise 2: Determine optimal hyperparameters based on 10-fold cross validation

In this exercise, you need to write code that will use your estimators from above to automatically choose the best hyperparameters for the histogram and kernel density estimator. In particular, find the best `n_bins` and `bandwidth` for the histogram and KDE respectively.

Task 1: Implement custom scorer function for use in GridSearchCV

To do this, you will need to implement a `scorer` function that will compute the log likelihood of the data given (higher is better). This function takes in the model, the input data X and y_true (which defaults to None since this is an unsupervised problem).

In [551]:
```
from sklearn.metrics import fbeta_score, make_scorer, log_loss
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import LeaveOneOut

def mean_log_likelihood_scorer(model, X, y_true=None):
    ########## Your code here ########
    # Compute and return the mean log probability of the data
    #  (Note y_true is not used)
    # scorer = make_scorer(neg_log_loss,true, true)
    epsi = 0.00000000000000000001

    pdf_vals = scipy.stats.norm.pdf(X).ravel()
    pdf_vals[pdf_vals < epsi] = epsi

    logs = np.log(pdf_vals)
    mean_ll = logs.mean()

    return mean_ll
    #############
```

Task 2: Estimate best hyperparameters

Then you can use sklearn's cross validation utilities to cross validate using the training data to determine the best parameters by passing this function as the `scoring` argument of GridSearchCV (note you just pass it directly as `mean_log_likelihood_scorer` without the parenthesis; this is known as passing a function to another function).

You should try 2-20 number of bins and a 50 bandwidth parameters linearly spaced between 0.1 and 10.

Finally, print out the optimal hyperparameters and, using the optimal hyperparameters, print out the log likelihood of the test data for both the histogram and KDE model.

In [542]:
```
def make_data(N, f=0.3, rseed=1):
    rand = np.random.RandomState(rseed)
    x = rand.randn(N)
    x[int(f * N):] += 5
    return x

X = make_data(1000).reshape(-1,1)
print(X.shape)
```

```
(1000, 1)
```

In [543]:
```
X = diff * np.vstack([scipy.stats.beta(6,1).rvs(size=(300,1)), scipy.stats.bet
a(2,7).rvs(size=(100,1))]) - diff/2
print(X.shape)
```

```
(400, 1)
```

In [552]:
```
############ Your code here #############3

min_val, max_val = 2,20

scorer = make_scorer(mean_log_likelihood_scorer, greater_is_better=True,y_true
= None)

n_bins = np.linspace(2, 20, 19).astype(int)


bandwidth = 10 ** np.linspace(0.1, 1, 100)

grid_hist = GridSearchCV(HistogramDensity(15,min_val= min_val, max_val=max_val
),{'n_bins' : n_bins},scoring= mean_log_likelihood_scorer,cv = 10, iid= True)

grid_ker = GridSearchCV(KernelDensity(0),{'bandwidth': bandwidth}, scoring= me
an_log_likelihood_scorer,cv = 10, iid= True)

#scorer = make_scorer(mean_log_likelihood_scorer,greater_is_better=True)
result_hist = grid_hist.fit(X);
result_ker = grid_ker.fit(X);
#grid_hist.fit(X)
#grid_ker.fit(X)
#results = CV_parameters.fit(X)

print(f'The optimal parameters are {grid_hist.best_params_, grid_ker.best_para
ms_ }')


print(f'The log likelihood probabilities are (Hist = {np.mean(result_hist.pred
ict_proba(X))[0]} KDE = {result_ker.predict_proba(X).mean() })')
```

```
The optimal parameters are ({'n_bins': 2}, {'bandwidth': 1.2589254117941673})
The log likelihood probabilities are (Hist = 0.05025069444444428 KDE = 0.1267
8221451295937)

D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:72: VisibleDeprecationWa
rning: Creating an ndarray from ragged nested sequences (which is a list-or-t
uple of lists-or-tuples-or ndarrays with different lengths or shapes) is depr
ecated. If you meant to do this, you must specify 'dtype=object' when creatin
g the ndarray
```