Saad Mufti

CS 539 - HW 5

Conceptual and Theoretical Questions

Q1 (10 pts)

Kullback-Leibler Divergence is given as:

$$KL(p||q) = -\int p(x) \ln\left(\frac{p(x)}{q(x)}\right)$$

And the PDF of a Multivariate Gaussian N is:

$$N_{PDF} = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Notice that the KL Divergence is also equivalent to

$$E_p\left[\frac{\ln(p)}{\ln(q)}\right]$$

Using the definition of expectation.

Compute to get:

$$= \frac{1}{2}\ln\frac{|L|}{|\Sigma|} - \frac{1}{2}E\left[(x-\mu)\Sigma^{-1}(x-\mu)^T\right] + \frac{1}{2}E\left[(x-m)^TL^{-1}(x-m)\right]$$

$$KL(p||q) = \frac{1}{2}\left(tr(L^{-1}\Sigma) + (m-\mu)^TL^{-1}(m-\mu) - 2 + \ln\frac{|L^{-1}|}{|\Sigma^{-1}|}\right)$$

Where k is the dimensionality of the Gaussians.

Q2 (10 pts)

Take the KL Divergence formula, replacing q with the normal PDF:

$$KL(p||N(x|\mu,\Sigma)) = -\int p(x)\ln\frac{p(x)}{\det(2\pi\Sigma)^{-\frac{1}{2}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$$

Take the partial derivatives with respect to Σ and μ and set them to 0 to find their minimizing expressions.

Take the partial with respect to μ:

$$\frac{\partial}{\partial\mu}p(x)\ln\left(\det(2\pi\Sigma)^{-\frac{1}{2}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)\right)$$
$$0 = \Sigma^{-1}\mu - \Sigma^{-1}E_p[x]$$
$$E_p[x] = \mu$$

And with respect to Σ:

$$\frac{\partial}{\partial\Sigma}p(x)\ln\left(\det(2\pi\Sigma)^{-\frac{1}{2}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)\right)$$
$$\Sigma = E[xx^T] - \mu\mu^T = E_p[xx^T] - E_p[x]E_p[x^T] = cov_p[x]$$

Thus, the optimal Normal distribution to approximate some other distribution, which would minimize their KL divergence, would be the Normal whose mean is its target's expectation and whose covariance is that of its target.

Q3 (10 pts)

Rewrite $p(t|x,w)$ as:

$$\prod_i^N \det(2\pi\Sigma)^{-\frac{1}{2}}\exp\left(-\frac{1}{2}(t_i - y(x_i,w))^T\Sigma^{-1}(t_i-y(x_i,w))\right)$$

Which is our likelihood function.

Simplify with the log-likelihood:

$$\propto \sum_i^N \ln\det(\Sigma)^{-\frac{1}{2}} - \frac{1}{2}(t_i - y(x_i,w))^T\Sigma^{-1}(t_i-y(x_i,w))$$
$$= -\frac{1}{2}\sum_i^N \ln\det(\Sigma) + (t_i - y(x_i,w))^T\Sigma^{-1}(t_i-y(x_i,w))$$

If Σ is fixed and known, we can remove the first term in the above equation to get:

$$\frac{1}{2}\sum_i^N (t_i - y(x_i,w))^T\Sigma^{-1}(t_i-y(x_i,w))$$

To find the maximum likelihood expression for Σ, take the derivative of the error function with respect to Σ, set it to 0, and solve for Σ.

Q4 (10 pts)

Rewrite z as $z = e^u$ to simplify further calculations, since $x^2 = uv$ and $x^2y = u$.

$$\frac{\partial z}{\partial u} = \frac{\partial}{\partial u} e^u = e^u$$
$$\frac{\partial z}{\partial v} = \frac{\partial}{\partial v} e^u = 0$$

If u and v are the tunable parameters for z, their update rules would be

$$u^{(n+1)} \leftarrow u^{(n)} - \eta \frac{\partial E}{\partial u}$$
$$v^{(n+1)} \leftarrow v^{(n)} - \eta \frac{\partial E}{\partial v}$$

Calculate the derivatives:

$$\frac{\partial E}{\partial u} = -\frac{\partial z}{\partial u} \cdot (t - e^u) = -e^u \cdot (t - e^u)$$
$$\frac{\partial E}{\partial v} = -\frac{\partial z}{\partial v} \cdot (t - e^u) = 0$$

Since E is not a function of v, its update rule is meaningless since changing v will not affect the error function. That is also why $\frac{\partial E}{\partial v} = 0$.

The update rule for u would become:

$$u^{(n+1)} \leftarrow u^{(n)} + \eta e^u (t - e^u)$$

Note I use E as the error function to avoid confusion with euler's constant.

Application Questions

In [ ]:
```
# Imports
from numpy import *
import pandas as pd
from scipy import integrate
from scipy.stats import norm

from sklearn.manifold import TSNE
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import cross_val_score

import matplotlib.pyplot as plt
```

KL Distance (20 pts)

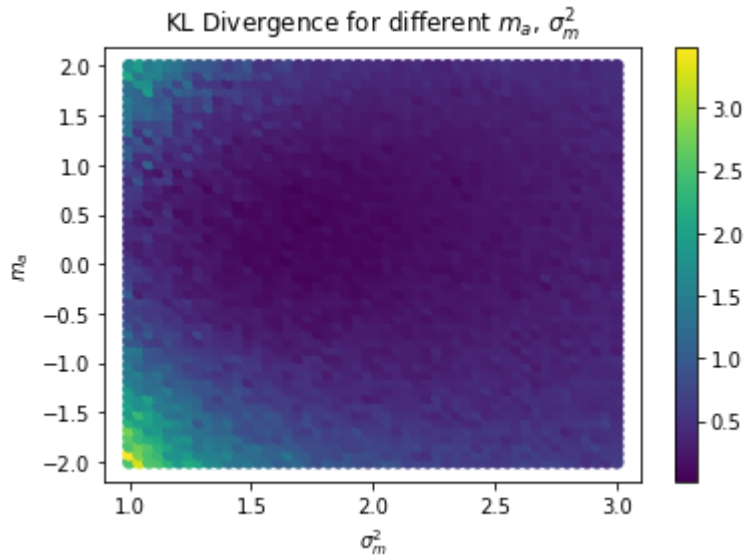
In [ ]:

```
# Part a
n_sam = 200
def p(x):
    return 1/3 * norm(-1, 2).pdf(x) + 2/3 * norm(1, 1).pdf(x)
def q(x, m, s):
    return norm(m, s).pdf(x)

def sample_p():
    samples = []
    sels = random.choice([0, 1], p=[1/3, 2/3], size=n_sam)
    for i in sels:
        mu, var = (-1, 2) if i == 0 else (1, 1)
        samples.append(random.normal(mu, var))
    return samples

def kl_distance(m, s, q):
    def eq(x):
        return log(p(x)) - log(q(x, m, s))
    return sum(eq(sample_p())) / n_sam

s_range = linspace(1, 3, num=50)
m_range = linspace(-2, 2, num=50)

dists = zeros((len(m_range), len(s_range)))

for i, m in enumerate(m_range):
    for j, s in enumerate(s_range):
        dists[i, j] = kl_distance(m, s, q)

# print(dists)
print("Mean of computed KL distances: ", round(dists.mean(), 5))
print(f"Min = {round(dists.min(), 5)}, Max = {round(dists.max(), 5)}")
```

```
Mean of computed distances:  0.43893
Min = 0.0126, Max = 3.49111
```

In [ ]:
```
fig_kl_a, ax_kl_a = plt.subplots()

X_s, Y_m = meshgrid(s_range, m_range)

scatter = ax_kl_a.scatter(X_s, Y_m, c=dists)
ax_kl_a.set_ylabel(r"$m_a$")
ax_kl_a.set_xlabel(r"$\sigma^2_m$")
ax_kl_a.set_title(r"KL Divergence for different $m_a$, $\sigma^2_m$")
fig_kl_a.colorbar(scatter, ax=ax_kl_a)

min_dist = round(dists.min(), 5)
arg_md = unravel_index(argmin(dists, axis=None), dists.shape)
est_min_m = Y_m[arg_md]
est_min_s = X_s[arg_md]
print(f"Min KL = {min_dist}, with:\nm = {round(est_min_m, 4)},\ns = {round(est
_min_s, 4)}")
```

```
Min KL = 0.0126, with:
m = 0.3673,
s = 1.6939
```

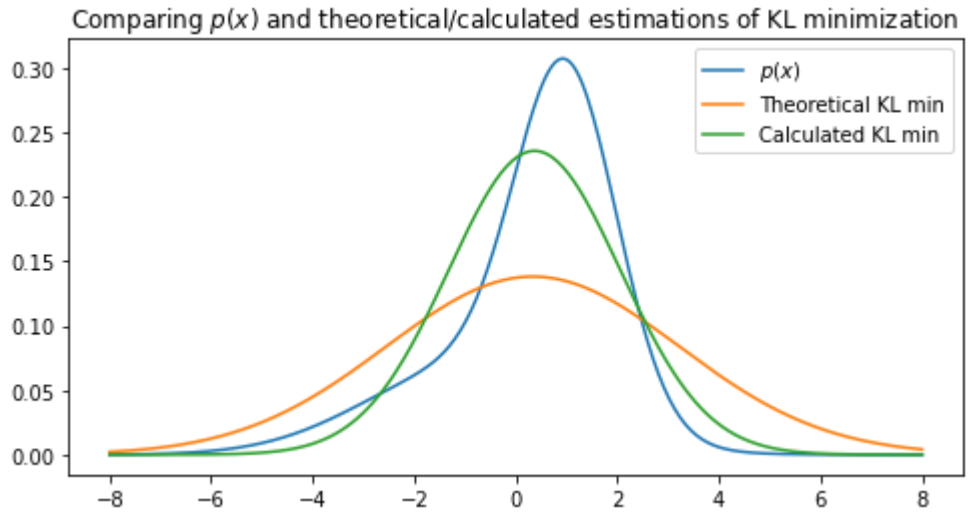


In [ ]:
```
# Part b
exp_p = integrate.quad(lambda x: x * p(x), -inf, inf)[0]
var_p = integrate.quad(lambda x: (x ** 2) * p(x), -inf, inf)[0] - exp_p ** 2

print("Expectation of p(x): ", round(exp_p, 5))
print("Variance of p(x): ", round(var_p, 5))
print("KL(p || q(E[p], Var[p]) = ", round(kl_distance(exp_p, var_p, q), 5))
```

```
Expectation of p(x):  0.33333
Variance of p(x):  2.88889
KL(p || q(E[p], Var[p]) =  0.26308
```

In [ ]:
```
fig_dist_comp, ax_dist_comp = plt.subplots(figsize=(8,4))

x_space = linspace(-8, 8, num=300)
ax_dist_comp.plot(x_space, p(x_space), label="$p(x)$")
ax_dist_comp.plot(x_space, q(x_space, exp_p, var_p), label="Theoretical KL mi
n")
ax_dist_comp.plot(x_space, q(x_space, est_min_m, est_min_s), label="Calculated
KL min")
ax_dist_comp.set_title("Comparing $p(x)$ and theoretical/calculated estimation
s of KL minimization")
ax_dist_comp.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x181a12a3760>



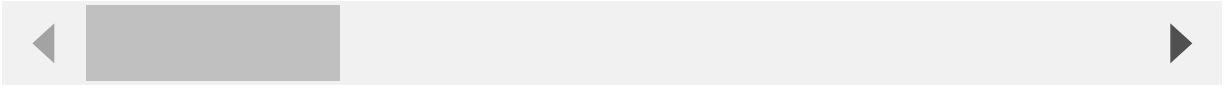
TSNE (20 pts)

In [ ]:
```
# Load training set predictors
ds_root = r"UCI HAR Dataset/UCI HAR Dataset"
features = pd.read_csv(f"{ds_root}/features_dup.txt", delimiter=" ", header=No
ne, names=["feature"])
features = features.to_numpy().T.flatten()
x_train = pd.read_csv(f"{ds_root}/train/X_train_pp.txt", delimiter=" ", header
=None, names=features)

x_train.head()
```

Out[ ]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672

5 rows × 561 columns



In [ ]:
```
# Load training set labels
y_train = pd.read_csv(f"{ds_root}/train/y_train.txt", header=None, names=["Lab
els"])
y_train = y_train.to_numpy().T.flatten()
y_train
```

Out[ ]:
```
array([5, 5, 5, ..., 2, 2, 2], dtype=int64)
```

In [ ]:
```
# Part a
tsne = TSNE(n_components=2, learning_rate="auto", init="pca")
x_train_red = tsne.fit_transform(x_train)
```

```
C:\Python39\lib\site-packages\sklearn\manifold\_t_sne.py:982: FutureWarning:
The PCA initialization in TSNE will change to have the standard deviation of
PC1 equal to 1e-4 in 1.2. This will ensure better convergence.
  warnings.warn(
```

In [ ]:
```
_, ax_tsne_a = plt.subplots()
ax_tsne_a.set_title("TSNE Representation of Training Set")
ax_tsne_a.scatter(*x_train_red.T, c=y_train)
```

```
7352, 7352
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x2b618ee6dd0>
```

In [ ]:
```
# Part b - Load testing set
x_test = pd.read_csv(f"{ds_root}/test/X_test_pp.txt", delimiter=" ", header=No
ne, names=features)
x_test.head()
```

Out[ ]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y
0	0.257178	-0.023285	-0.014654	-0.938404	-0.920091	-0.667683	-0.952501	-0.925249
1	0.286027	-0.013163	-0.119083	-0.975415	-0.967458	-0.944958	-0.986799	-0.968401
2	0.275485	-0.026050	-0.118152	-0.993819	-0.969926	-0.962748	-0.994403	-0.970735
3	0.270298	-0.032614	-0.117520	-0.994743	-0.973268	-0.967091	-0.995274	-0.974471
4	0.274833	-0.027848	-0.129527	-0.993852	-0.967445	-0.978295	-0.994111	-0.965953

5 rows × 561 columns

In [ ]:
```
# Load testing labels
y_test = pd.read_csv(f"{ds_root}/test/y_test.txt", header=None, names=["Label
s"])
y_test = y_test.to_numpy().T.flatten()
y_test
print(y_test.shape)
```

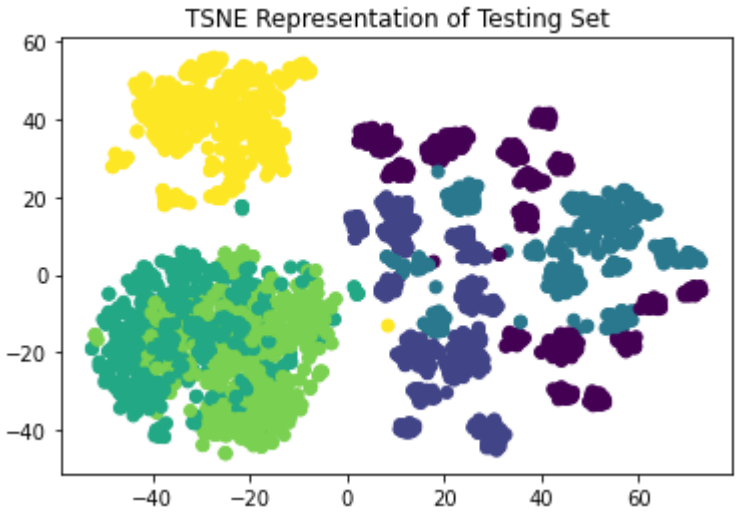
```
(2947,)
```

In [ ]:
```
# Perform t-SNE on testing set
x_test_red = tsne.fit_transform(x_test)
```

```
C:\Users\Saad\AppData\Local\Programs\Python\Python310\lib\site-packages\sklea
rn\manifold\_t_sne.py:982: FutureWarning: The PCA initialization in TSNE will
change to have the standard deviation of PC1 equal to 1e-4 in 1.2. This will
ensure better convergence.
  warnings.warn(
```

In [ ]:
```
_, ax_tsne_b = plt.subplots()
ax_tsne_b.set_title("TSNE Representation of Testing Set")
ax_tsne_b.scatter(*x_test_red.T, c=y_test)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x2b618f9c100>
```



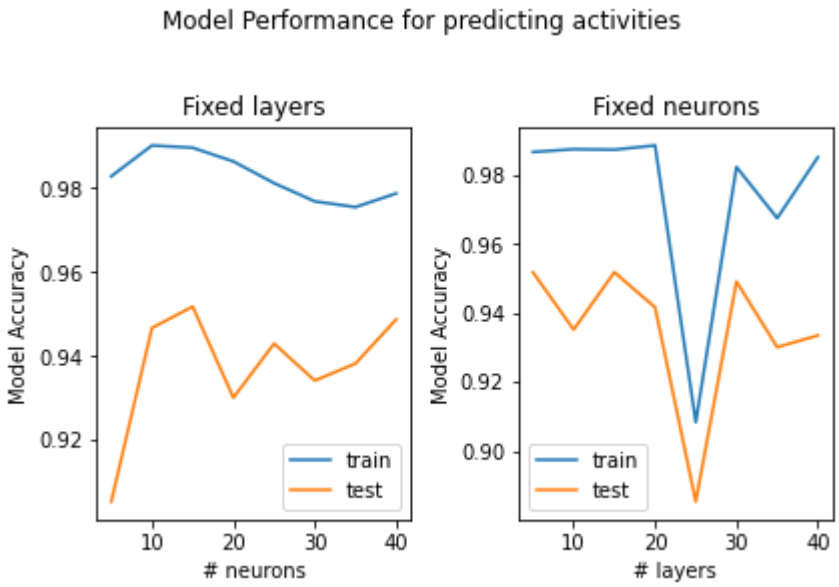
Part c

The test and training sets bear some similarities, such as around the yellow and greensish data on the left side of the plots. The yellow clusters are relatively far from other groups of points, although the test set's yellow cluster is less consolidated. The same goes for the mix of greenish clusters overlayed on each other, which are roughly the same in both datasets. The turquoise data in the training set are close together, while in the test set, exist in small clusters.

Overall, the similarities between the test and training set are apparent, but there are some notable differences that set them apart.

Neural Networks (20 pts)

In [ ]:
```
# Part a

fig_perf_a, ax_perf_a = plt.subplots(1, 2)
def nn_perf(l_train, l_test, ax):
    x_space = [5, 10, 15, 20, 25, 30, 35, 40]
    for f in [0, 1]:
        acc_train = []
        acc_test = []
        for i in x_space:
            layers = (i,) * 20 if f == 0 else (20,) * i
            model = MLPClassifier(hidden_layer_sizes=layers, activation="ident
ity")
            model.fit(x_train, l_train)
            acc_train.append(model.score(x_train, l_train))
            acc_test.append(model.score(x_test, l_test))
        ax[f].plot(x_space, acc_train, label="train")
        ax[f].plot(x_space, acc_test, label="test")
        ax[f].set_ylabel("Model Accuracy")
        ax[f].legend()

nn_perf(y_train, y_test, ax_perf_a)
fig_perf_a.suptitle("Model Performance for predicting activities", y=1.03)
ax_perf_a[0].set_title("Fixed layers")
ax_perf_a[1].set_title("Fixed neurons")
ax_perf_a[0].set_xlabel("# neurons")
ax_perf_a[1].set_xlabel("# layers")
fig_perf_a.tight_layout()
```



In [ ]:
```
# Logistic Regression (with no regularization) also performs very well in pred
icting activities
from sklearn.linear_model import LogisticRegression
lin = LogisticRegression(penalty="none", max_iter=2000).fit(x_train, y_train)
lin.score(x_test, y_test)
```

Out[ ]: 0.9504580929759077

Observations

This dataset behaved very well overall. The graph above indicates some different neural network structures tested on the the test set, which all worked with <=90% accuracy. The identity activation function worked best (all others offered from the sklearn library were significantly worse), which made sense since the data is actually linearly separable, which I proved by running a Logistic Regression model on the data, which yielded ~95% accuracy on the test set. My experimentation with the NN structure was not exhaustive, but I found that the most optimal configuration was with 15 layers, with 20 neurons per layer.

In [ ]:
```
# Part b - Load participant data
particip_train = pd.read_csv(f"{ds_root}/train/subject_train.txt", header=None
)
particip_train = particip_train.to_numpy().T.flatten()

particip_test = pd.read_csv(f"{ds_root}/test/subject_test.txt", header=None)
particip_test = particip_test.to_numpy().T.flatten()
```

In [ ]:
```
# Combine test/train datasets
X = pd.concat([x_train, x_test])
particip_set = concatenate([particip_train, particip_test])
model_b = MLPClassifier(hidden_layer_sizes=(15,)*2, activation="identity")

cross_val_score(model_b, X, particip_set, cv=10)
```

```
C:\Python39\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.p
y:699: UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Python39\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.p
y:692: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) rea
ched and the optimization hasn't converged yet.
  warnings.warn(
C:\Python39\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.p
y:699: UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Python39\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.p
y:699: UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Python39\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.p
y:692: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) rea
ched and the optimization hasn't converged yet.
  warnings.warn(
C:\Python39\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.p
y:692: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) rea
ched and the optimization hasn't converged yet.
  warnings.warn(
C:\Python39\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.p
y:692: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) rea
ched and the optimization hasn't converged yet.
  warnings.warn(
```

Out[ ]:
```
array([0.22135922, 0.27961165, 0.47961165, 0.68543689, 0.71067961,
       0.32330097, 0.24854369, 0.49514563, 0.76601942, 0.6404276 ])
```

Observations The subject dataset, on the other hand, was not as simple to train on. Because the test set was comprised of labels the training set did not have, the NN classifier initially had an accuracy of 0, which is why it was necessary to recombine the test and training sets to show the neural network at least some examples of each subject.

The 10-fold cross validation for my chosen model did not give nearly as good accuracies as compared to a). This was the best accuracy I could produce for the models I tried.

My activation function was once again the identity function, and I used 15 neurons in each of 2 layers. I previously tried a larger number of layers, but with worse results.

The accuracy of the folds ranges from 20% to nearly 80%.