Midterm 1

This exam has three questions. The first problem has several True/False questions, covering various concepts we've covered so far. The next two problems ask you to implement different models on some small data sets, and to answer questions about the results.

You can use any online resource; this is intended to allow to you to access documentation and examples for coding as you would when you program in practice. *You are expressly forbidden to have any communication with another person--either inside or outside of the class--during the exam.*

If you get stuck on a coding part, please include comments or markdown describing how you would go about getting a working solution. We will take this into account to assign partial credit.

- You have 1 hour and 15 minutes to complete the exam.
- When you are finished, upload to Canvas your notebook and pdf (from html) printout, just as you do for the assignments.
- Pass in the signature and grading sheet we are providing before you leave OML 202.

Problem 1: True/False (15 points)

Indicate your answer for each question by replacing [TF] with [T] for True and replacing [TF] with [F] for False.

1.1 Classification and regression

Consider a classification task where the input is d-dimensional and the output Y is binary

1. A method is overfitting the data when it has a small training error but a large test error. [TF]

In [4]:
```
# TRUE
```

1. Logistic regression is more accurate than linear discriminant analysis because logistic regression does not give a linear decision boundary. [TF]

In [177]:
```
# FALSE: LR has a linear boundary
```

1. Leave-one-out cross validation gives a more accurate estimate of test error than R^2. [TF]

In [178]:
```
# TRUE
```

1. K-nearest neighbors regression will always give a better fit than linear regression [TF]

In [179]:
```
# FALSE
```

1. If X is an $n \times p$ data matrix, where each column mean is zero, then $\frac{1}{n} X^T X$ is the sample covariance matrix. [TF]

In [180]:
```
# TRUE
```

1.2 Trees and random forests

Consider growing random forests for regression. We can vary the depth (or "height") h of the tree or the number of random predictors m chosen for each split.

1. Each bootstrap sample of a data set contains about 63% of the original data. [TF]

In [185]:
```
# TRUE
```

1. When we penalize a method, the variance increases. [TF]

In [184]:
```
# FALSE
```

1. Pruning a decision tree helps to decrease the bias of the model. [TF]

In [181]:
```
# FALSE: it helps reduce variance of a deep tree
```

1. Random forest regression will give the same result whether or not the data is standardized before fitting. [TF]

In [183]:
```
# TRUE
```

1. Bootstrap aggregation and random forests are ensemble methods. [TF]

In [6]:
```
# TRUE
```

1.3 PCA and gradient descent

1. Stochastic gradient descent optimizes a loss function using random samples of the data at each step to optimize a model's parameters. [TF]

In [188]:
```
# FALSE
```

1. Stochastic gradient descent can be used to maximize the probability that a logistic regression model assigns to the labels in the training data. [TF]

In [187]:
```
# TRUE
```

1. Applying gradient descent to the loss function for logistic regression should give approximately the same answer for any initialization of the parameters. [TF]

In [186]:
```
# TRUE
```

1. PCA finds directions of greatest variation in the data. [TF]

In [7]:
```
# TRUE
```

1. The first and second principal vectors from PCA are always orthogonal to each other. [TF]

In [8]:
```
# TRUE
```

Problem 2: Logistic regression (20 points)

Have you every gone for a walk in the woods and been tempted to bring back some mushrooms for your dinner salad? Well, even if not, it makes sense to ask how many mushrooms we would need to see in order to accurately classify a mushroom as edible or poisonous. You'll get an answer in this problem.

We'll first load a database of mushrooms that have been hand-classified according to whether or not they are poisonous. Then, we'll put it into a form suitable for logistic regression by converting all of the categorical variables to "dummy" variables using a "1-hot" representation. Then you'll fit logistic regression models on training sets of increasing size.

In [6]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression

import matplotlib.pyplot as plt
%matplotlib inline
```

In [7]:
```
data = pd.read_csv('mushrooms.csv')
data.head()
```

Out[7]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...
0	p	x	s	n	t	p	f	c	n	k	...
1	e	x	s	y	t	a	f	c	b	k	...
2	e	b	s	w	t	l	f	c	b	n	...
3	p	x	y	w	t	p	f	c	n	n	...
4	e	x	s	g	f	n	f	w	b	k	...

5 rows × 23 columns

In [8]:
```
data['class'] = LabelEncoder().fit_transform(data['class'])
encoded_data = pd.get_dummies(data)
encoded_data.head()
```

Out[8]:

	class	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap surface_(
0	1	0	0	0	0	0	1	0	(
1	0	0	0	0	0	0	1	0	(
2	0	1	0	0	0	0	0	0	(
3	1	0	0	0	0	0	1	0	(
4	0	0	0	0	0	0	1	0	(

5 rows × 118 columns

2.1 Explain the transformation of the data

Explain in words what the transformations above are doing, to prepare the data for classification, and why the original data are not in a suitable form.

- One-hot encoding is applied to the letter representations. This is where the letter encoded variable is removed and a new binary variable is added for each unique integer value.
- This increases the number of "features" available for the logistic regression. The original data is not suitable because logistic regression internally assigns a weight to each feature. With a categorical variable (unordered), it cannot assign a weight to each choice. But one-hot encoding lets the regression deal with each category independently.

Next, we convert this DataFrame to numpy arrays, suitable for input to sklearn.

In [9]:
```
y = encoded_data['class'].values
X = encoded_data.drop('class', 1).values
```

In [15]:
```
len(y[y == 1])
```

Out[15]:
```
3916
```

2.2 Dimensions of the data

Ok, now before you get to work building a classifier, answer a couple of questions about the data. How many data points are there? How many original predictors? How many predictor variables after expansion using dummy variables? What percentage of the mushrooms in the data set are poisonous?

In [17]:
```
num_data_points = 8124
num_original_predictors = 22
num_expanded_predictors = 117
poisonous = (3916 / 8124) * 100

print("Number of data points: %d" % num_data_points)
print("Number of original predictors: %d" % num_original_predict
ors)
print("Number of expanded predictors: %d" % num_expanded_predict
ors)
print("Percent poisonous: %.1f%%" % poisonous)
```

```
Number of data points: 8124
Number of original predictors: 22
Number of expanded predictors: 117
Percent poisonous: 48.2%
```

2.3 Fitting logistic regression models

Next, you should construct a series of logistic regression models with an increasing number of training points. Specifically:

- let the sample size n vary from 2 to 400 in increments of 1
- for each n train a logistic regression model on n randomly selected training points, and test on the remaining data
- For each n, do this 10 times and average the error rates
- Plot the resulting average error rates as a function of n

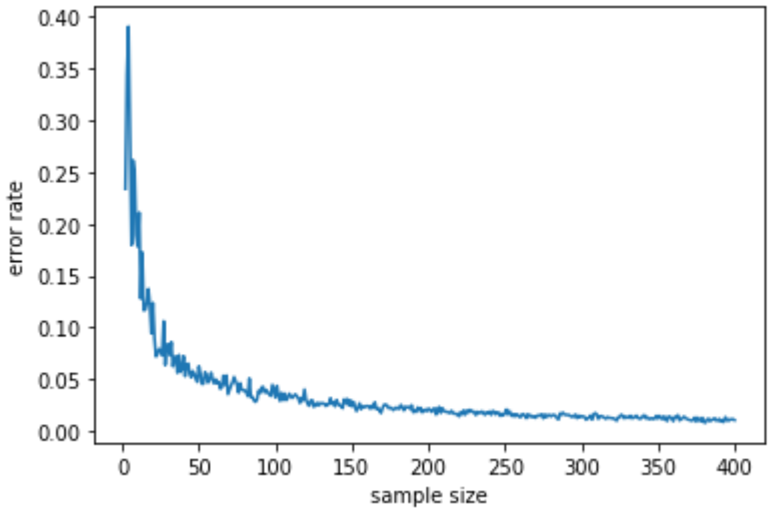
Some important notes and hints:

- Use the function `sklearn.model_selection.train_test_split` in each trial to randomly split the data into training and test sets. For example, you can use `train_test_split(X, y, train_size=n)`
- When n is small, the model will fail to train if all of the training examples have the same label. When this happens, just skip that data set and choose another random split.
- The implementation only requires about 10-15 lines of code. If you find yourself writing a lot more, reconsider your approach!

In [21]:
```
from sklearn.metrics import accuracy_score
trials = 10
sample_size = np.arange(2, 401, 1)
# error_rate = np.zeros(len(sample_size))
error_rate = []
lr = LogisticRegression(solver='lbfgs')
```

In [24]:
```
# your code here
for ss in sample_size:
    local_error = []
    for i in range(trials):
        X_train, X_test, y_train, y_test = train_test_split(X,
y, train_size=ss)
        try:
            lr.fit(X_train, y_train)
        except:
            continue
        y_pred = lr.predict(X_test)
        local_error.append(1 - accuracy_score(y_test, y_pred))
    error_rate.append(np.mean(local_error))
```

In [26]:
```
plt.plot(sample_size, error_rate)
plt.xlabel('sample size')
_ = plt.ylabel('error rate')
```



2.4 Sample sizes to get 5% and 1% error

Finally, answer the following questions about your results. How many data points are required before the error falls below 5 percent? Below 1 percent?

In [31]:
```
p5 = []
for err in error_rate:
    if err >= 0.05:
        p5.append(err)
```

In [32]:
```
len(p5)
```

Out[32]:
```
58
```

In [33]:
```
p1 = []
for err in error_rate:
    if err >= 0.01:
        p1.append(err)
```

In [34]:
```
len(p1)
```

Out[34]:
```
390
```

In [35]:
```
n_05 = 58 # your code here
print("%d samples are required before the error falls below 5%%"
% n_05)

n_01 = 390 # your code here
print("%d samples are required before the error falls below 1%%"
% n_01)
```

```
58 samples are required before the error falls below 5%
390 samples are required before the error falls below 1%
```

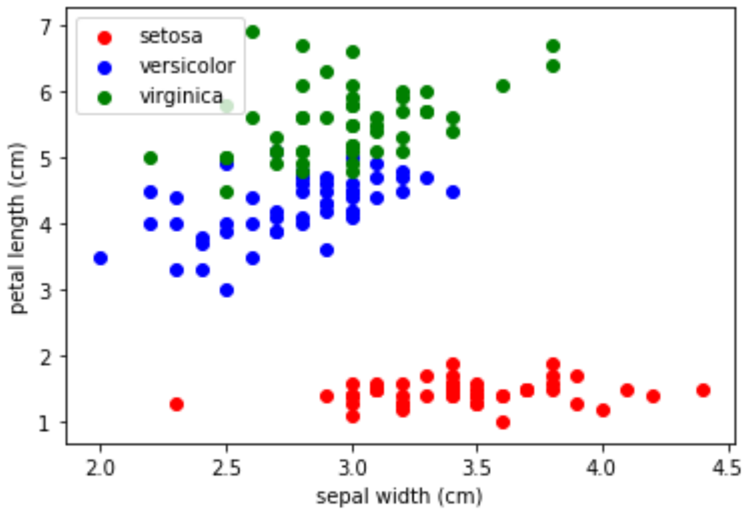
Problem 3: PCA and classification (30 points)

In this problem you will carry out principal components analysis and classification on the iris data. The task will be to reduce the dimension from four to two using PCA, and then to compare decision tree classifiers with logistic regression.

In [155]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt
%matplotlib inline
```

The following cell plots the original data in terms of sepal width and petal length.

In [156]:
```
# don't change this code
from sklearn.datasets import load_iris

fig = plt.figure()
iris = load_iris()
x_index = 1
y_index = 2
ax = fig.add_subplot(111)
colors = ['red', 'blue', 'green']
for c in np.arange(3):
    mask = (iris.target==c)
    plt.scatter(iris.data[mask, x_index], iris.data[mask, y_inde
x], color=colors[c], label=iris.target_names[c])

plt.xlabel(iris.feature_names[x_index])
plt.ylabel(iris.feature_names[y_index])
plt.legend(loc='upper left')
plt.show()
```



In [157]:
```
X = iris.data
y = iris.target
```

```
In [158]: iris.feature_names
```

```
Out[158]: ['sepal length (cm)',
           'sepal width (cm)',
           'petal length (cm)',
           'petal width (cm)']
```

3.1 Carry out PCA

In the following cell, reduce the dimension of the data from four to two dimensions by carrying out principal components analysis. Let `pv1` be the first principal vector and let `pv2` be the second principal vector. Let `pcs` be the principal components, that is, the projection of the data onto the first two principal vectors.

```
In [159]: pca = PCA(n_components=2)
          X = X - np.mean(X,0)
```

```
In [160]: pcs = pca.fit_transform(X) # the principal components
          principal_vectors = pca.components_ # the principal vectors
          pv1 = principal_vectors[0]
          pv2 = principal_vectors[1]
```

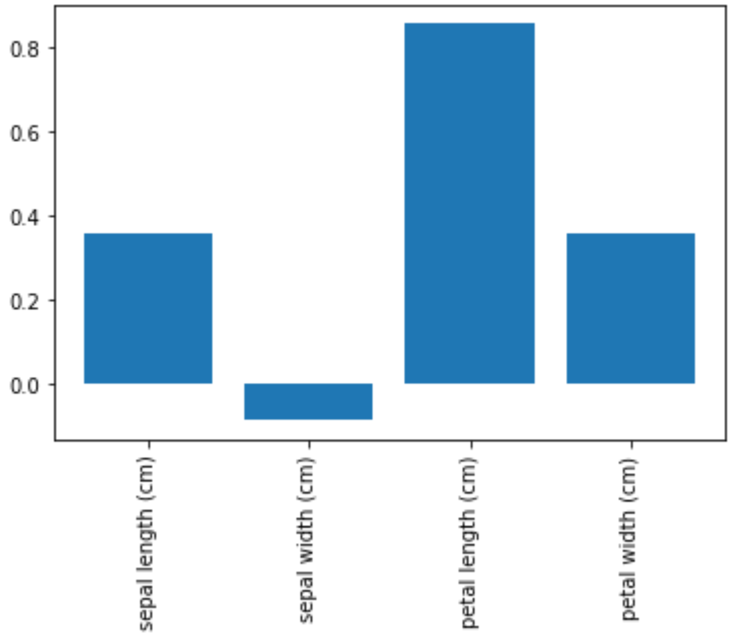
```
In [161]: principal_vectors
```

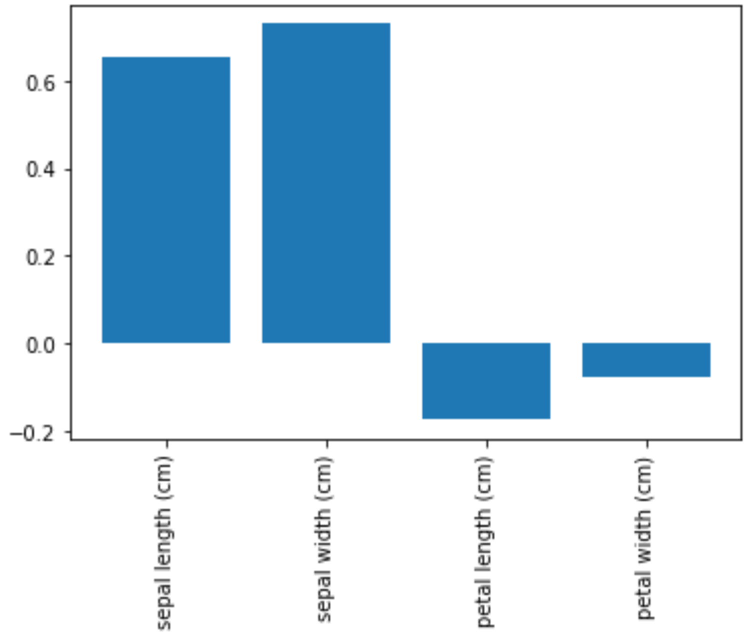
```
Out[161]: array([[ 0.36138659, -0.08452251,  0.85667061,  0.3582892 ],
                 [ 0.65658877,  0.73016143, -0.17337266, -0.07548102]])
```

```
In [162]: # Comment out the following lines!
          # pv1 = np.random.normal(size=4)
          # pv2 = np.random.normal(size=4)
          # pcs = np.random.normal(size=2*X.shape[0]).reshape(X.shape[0],
          2)
```

The next cell plots the principal vectors. (Don't change this code.)

In [163]:
```
plt.bar(np.arange(4), pv1)
plt.xticks(np.arange(4), iris.feature_names, rotation='vertical
')
plt.show()

plt.bar(np.arange(4), pv2)
plt.xticks(np.arange(4), iris.feature_names, rotation='vertical
')
plt.show()
```





3.2 Qualitative description of vectors

Describe qualitatively the main properties of the data that the first two principal vectors are capturing.

- The first vector shows the flowers most important in their petal length.
- The second vector shows the flowers most important in their sepal length and sepal width.

3.3 Verify orthogonality

Write a single code that evaluates to `True` if the first two principal vectors are orthogonal, and to `False` otherwise.

```
In [164]: pv1
```

```
Out[164]: array([ 0.36138659, -0.08452251,  0.85667061,  0.3582892 ])
```

```
In [165]: pv2
```

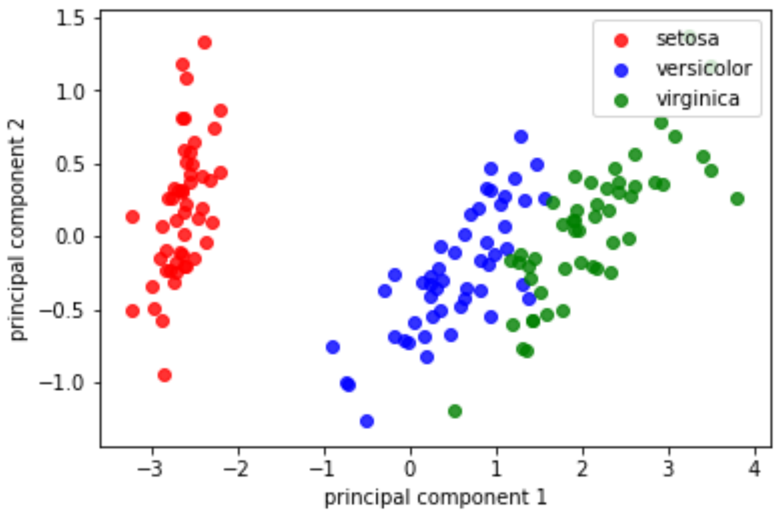
```
Out[165]: array([ 0.65658877,  0.73016143, -0.17337266, -0.07548102])
```

```
In [166]: # floating point arithmetic is not completely accurate, so it's
          fine
          np.dot(pv1, pv2)
```

```
Out[166]: 3.885780586188048e-16
```

Finally, plot the projections of data onto the first two principal components. To do this, use the code provided below, where the principal components `pcs` are the values you computed above. (Don't change the code in the following cell.)

In [167]:
```
# do not change this code

fig = plt.figure()
ax = fig.add_subplot(111)
colors = ['red', 'blue', 'green']
for i in np.arange(3):
    mask = y==i
    ax.scatter(pcs[mask,0], pcs[mask,1], alpha=0.8, c=colors[i],
label=iris.target_names[i])
plt.legend(loc='upper right')
plt.xlabel('principal component 1')
plt.ylabel('principal component 2')
plt.show()
```



3.4 Describe your plot

Describe the resulting plot. How is it different than and similar to the plot above for petal length vs. sepal width?

- It is similar in the sense that versicolor and virginica have some quality which makes them similar and their data points partially overlap.
- It is also similar in that setosa is clearly distinguished from the other two.
- However, there is greater variability in the data points. In the first plot, many of them share the same sepal width etc. and therefore they are less dispersed. Here the new components either capture smaller differences between the points or the perturbations could be due to reconstruction error.

3.5 Fitting decision trees

Next, you construct a series of decision tree classifiers **using the first two principal components** as predictor variables, with an increasing number of training points. Specifically:

- let the sample size vary between 10% of the data to 90% of the data, in increments of 10%
- for each sample size, train a decision tree on randomly selected training points, and test on the remaining data
- For each sample size, run 1000 trials and average the error rates
- Plot the resulting average error rates as a function of sample percentage of the data

If you are not confident that you have the correct principal components, you may use the petal length and sepal width variables for partial credit.

Some important notes and hints:

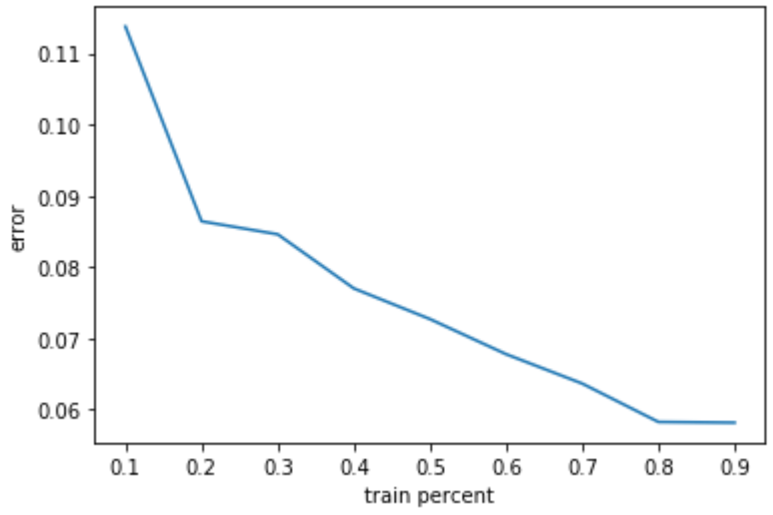
- Use the function `sklearn.model_selection.train_test_split` in each trial to randomly split the data into training and test sets. For example, you can use `train_test_split(X, y, train_size=.1)`
- The implementation only requires about 10-15 lines of code. If you find yourself writing a lot more, reconsider your approach!

In [168]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

In [169]:
```
X = pcs
dtree = DecisionTreeClassifier()
train_percent = np.linspace(.1,.9,num=9)
# dt_error_rate = np.zeros(len(train_percent))
dt_error_rate = []
trials = 1000
```

In [170]:
```
for p in train_percent:
    errs = []
    for trial in np.arange(trials):
        X_train, X_test, y_train, y_test = train_test_split(X,
y, train_size=p)
        dtree.fit(X_train, y_train)
        err = np.mean(dtree.predict(X_test) != y_test)
        errs.append(err)
    this_err = np.mean(errs)
    dt_error_rate.append(this_err)
```

In [171]:
```
plt.plot(train_percent, dt_error_rate)
plt.xlabel('train percent')
_ = plt.ylabel('error')
```



3.6 Finding the minimum error

What is the smallest average error attained by the decision trees?

In [172]:
```
min_error = min(dt_error_rate)
print("The minimum error attained by the decision trees is %.2
f%%" % (100*min_error))
```

```
The minimum error attained by the decision trees is 5.81%
```

3.7 Interpreting the trees

Note that the decision trees are classifying into three types, namely the three species *virginica*, *setosa*, and *versicolor*. Explain in words how a decision tree makes this prediction at each leaf in the tree.

- A data point traverses the tree, answering a series of questions.
- The outcome of the PCA was numeric, so there are numerous splits along the tree which minimise the error metric (i.e. MSE).
- The data point ends up at the leaf according to how it performed at each split.

3.8 Training logistic regression models

Next, construct a series of logistic regression using an increasing number of training points, **also using the first two principal components as predictor variables**. Specifically:

- let the sample size 10% of the data to 90% of the data, in increments of 10%
- for each sample size, train a logistic regression model on randomly selected training points, and test on the remaining data
- For each sample size, run 1000 trials and average the error rates
- Plot the resulting average error rates as a function of sample percentage of the data

As above, if you are not confident that you have the correct principal components, you may use the petal length and sepal width variables for partial credit.

Some important notes and hints:

- As before, use the function sklearn.model_selection.train_test_split in each trial to randomly split the data into training and test sets. For example, you can use train_test_split(X, y, train_size=.1)
- Use `lr = LogisticRegression(solver='lbfgs', multi_class='multinomial')` as written. This will fit a logistic regression model to predict the three class labels (*versicolor*, *virginica*, and *setosa*). This is a linear model of the log-odds, just as for binary logistic regression. The decision boundaries will be linear functions of the two principal components.
- In a little more detail, in a logistic regression model to carry out three-way classification where Y can be 0, 1, or 2, the model takes the form

$$P(Y = 0 \mid x) = \frac{1}{1 + e^{\beta_1^T x} + e^{\beta_2^T x}}$$

$$P(Y = 1 \mid x) = \frac{e^{\beta_1^T x}}{1 + e^{\beta_1^T x} + e^{\beta_2^T x}}$$

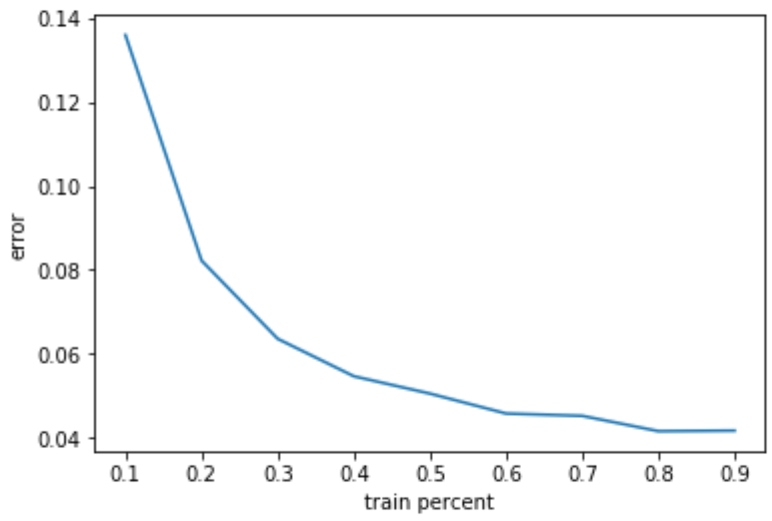
$$P(Y = 2 \mid x) = \frac{e^{\beta_2^T x}}{1 + e^{\beta_1^T x} + e^{\beta_2^T x}}$$

- The decision boundaries between any two classes in such a model are linear.

```
In [173]: from sklearn.linear_model import LogisticRegression
          from sklearn.model_selection import train_test_split

          X = pcs
          lr = LogisticRegression(solver='lbfgs', multi_class='multinomial
          ')
          train_percent = np.linspace(.1,.9,num=9)
          # lr_error_rate = np.zeros(len(train_percent))
          lr_error_rate = []
          trials = 1000
```

```
In [174]: for p in train_percent:
              errs = []
              for trial in np.arange(trials):
                  X_train, X_test, y_train, y_test = train_test_split(X,
          y, train_size=p)
                  lr.fit(X_train, y_train)
                  err = np.mean(lr.predict(X_test) != y_test)
                  errs.append(err)
              this_err = np.mean(errs)
              lr_error_rate.append(this_err)
```

```
In [175]: plt.plot(train_percent, lr_error_rate)
          plt.xlabel('train percent')
          _ = plt.ylabel('error')
```



3.9 Finding the minimum error

What is the smallest average error attained by logistic regression?

In [176]:
```
min_error = min(lr_error_rate) # your code here
print("The minimum error attained by the logistic regression is
%.2f%%" % (100*min_error))
```

```
The minimum error attained by the logistic regression is 4.16%
```

3.10 Explaining the results

Which performs better, decision trees or logistic regression? If one performs better than the other, explain why this is the case. If they perform about equally, also explain why this might be the case.

- In terms of minimum error, logistic regression is better, with 4.16% vs decision tree's 5.81%. However, this error is just in training. Both models could be overfitting, and we need to test on novel data.
- However, while decision tree error drops to about 0.085 by 20% training data, the logistic regression error drops quicker, to about 0.08 by 20% training data. This might indicate that is learning the underlying parameters better, with less data.
- Moreover, the logistic regression approach has a steeper slope throughout, whereas the dtree has almost linear decreases after 30% training data.