1) KNN

a) The answer is ii). We should calculate the mean and standard deviation of the training set, and apply those numbers based on the training set to both sets. We do not want to use the whole data (that includes the test set) to get the mean and standard deviation because that is introducing unseen information. We also don't want to separately use the mean and standard deviation from the test set because we don't want to create relationship between the test set. We should view it as we can only see one test data point at a time.

b) The value of k is k = 10. We can find counterexamples for k = 1 because there are blue points in the orange region and that area is all classified as orange, so definitely not 1. Similarly for k = 2, we can see there are groups of orange points, for example there are is a group of orange points in the blue area where the area is still all classified as blue. We find that for k = 10, there is no counterexample. Therefore, we conclude that k = 10. k = 100 is too large.

c) (Executing the command to get the data. Nothing to solve.)

d)

Question 2 : Choosing k by cross-validation

We have implemented the k-Nearest Neighbor classifier but we set the value k = 5 arbitrarily. We will now determine the best value of this hyperparameter with cross-validation. What is the best value for k ?

k=15

```
from sklearn.model_selection import train_test_split
num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

X_train_folds = []
y_train_folds = []
################################################################################
# TODO:                                                                        #
# Split up the training data into training and validation set. After splitting,#
# X_train_folds and y_train_folds should each be lists of length num_folds,    #
# where y_train_folds[i] is the label vector for the points in X_train_folds[i]#
# Hint: Look up the numpy array_split function.                                #
################################################################################
X_train_folds = np.array(np.array_split(X_train, num_folds))
y_train_folds = np.array(np.array_split(y_train, num_folds))
################################################################################
#                                END OF YOUR CODE                              #
################################################################################

# A dictionary holding the accuracies for different values of k that we find
# when running validation. After running validation, k_to_accuracies[k] shold be
# the accuracy for using k value for n_neighbors
k_to_accuracies = {}

################################################################################
# TODO:                                                                        #
# Perform validation to find the best value of k. For each                     #
# possible value of k, run the k-nearest-neighbor algorithm num_folds times,   #
# where in each case you use all but one of the folds as training data and the #
# last fold as a validation set. Store the accuracies for all fold and all     #
# values of k in the k_to_accuracies dictionary.                               #
# expect 8 lines of the code                                                   #
################################################################################
for k in k_choices:
    for fold in range(num_folds):
        comb = [x for x in range(num_folds) if x != fold]
        xres = np.concatenate(X_train_folds[comb])
        yres = np.concatenate(y_train_folds[comb])
        classifierk = KNeighborsClassifier(n_neighbors = k, n_jobs = 4)
        classifierk.fit(xres, yres)
        predictres = classifierk.predict(X_train_folds[fold])
        num_correct = np.sum(predictres == y_train_folds[fold])
        accuracy = float(num_correct) / num_test
        k_to_accuracies[k] = accuracy
```

```
# Based on the cross-validation results above, choose the best value for k,
# retrain the classifier using all the training data, and test it on the test
# data. You should be able to get above 28% accuracy on the test data.

X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))

# TODO :: use the best k to train and predict on the test set, expect 4 lines of code
classifier = KNeighborsClassifier(n_neighbors = 15, n_jobs = 4)
classifier.fit(X_train, y_train)
y_test_predict = classifier.predict(X_test)

# TODO :: Compute and display the accuracy, expect 3 lines of code
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('Got %d / %d correct => accuracy: %f' % (num_correct, num_test, accuracy))
```

2) Decision Tree
a)
cost(D) = 400/800 = 1/2

Model A:
cost(D_left) = 100/400 = 1/4
cost(D_right) = 100/400 = 1/4
Reduction(D) = 1/2 - (1/2*1/4+1/2*1/4) = 1/4

Model B:
cost(D_left) = 200/600 = 1/3
cost(D_right) = 0/200 = 0
Reduction(D) = 1/2 - (3/4*1/3 + 0) = 1/4

According to the calculations, both models are the same.

b)
Entropy

cost(D) = -(400/800)log_2(400/800)-(1-400/800)log_2(1-400/800) = 1

Model A:
cost(D_left) = -(300/400)log_2(300/400)-(100/400)log_2(100/400) = 0.811
cost(D_right) = -(100/400)log_2(100/400)-(300/400)log_2(300/400) = 0.811
Reduction(D) = 1 - (1/2*0.811+1/2*0.811) = 0.189

Model B:
cost(D_left) = -(200/600)log_2(200/600)-(400/600)log_2(400/600) = 0.918
cost(D_right) = -(200/200)log_2(200/200)-(0)log_2(0) = 0
Reduction(D) = 1-(600/800*0.918+0) = 0.3115

Preferred split is model B.

c)
Gini Index

cost(D) = 2(400/800)(400/800) = 1/2

Model A:
cost(D_left) = 2(300/400)(1-300/400) = 0.375
cost(D_right) = 2(100/400)(1-100/400) = 0.375
Reduction(D) = 1/2-(1/2*0.375+1/2*0.375) = 0.125

Model B:
cost(D_left) = 2(200/600)(1-200/600) = 0.444
cost(D_right) = 2(200/200)(1-200/200) = 0
Reduction(D) = 1/2-(600/800*0.444+0) = 0.167

Preferred split is model B.

d)

Question 1 Implement Decision Tree

In [41]:
```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
# TODO :: define a sklearn DecisionTreeClassifier and set the max_depth to 3, expect 1 line of code
decision_tree_binary_classifier = DecisionTreeClassifier(max_depth = 3)
# TODO :: fit the classifier your defined earlier, and fit on training data
decision_tree_binary_classifier.fit(X_train, y_train)
```

Out[41]:
```
DecisionTreeClassifier(max_depth=3)
```

In [42]:
```
import numpy as np
y_pred_train = decision_tree_binary_classifier.predict(X_train)
num_correct = np.sum(y_pred_train == y_train)
print("accuracy on test set : {}".format(num_correct / float(len(y_train))))
```

```
accuracy on test set : 1.0
```

In [43]:
```
y_pred_test = decision_tree_binary_classifier.predict(X_test)
num_correct = np.sum(y_pred_test == y_test)
print("accuracy on test set : {}".format(num_correct / float(len(y_pred_test))))
```

```
accuracy on test set : 0.6
```

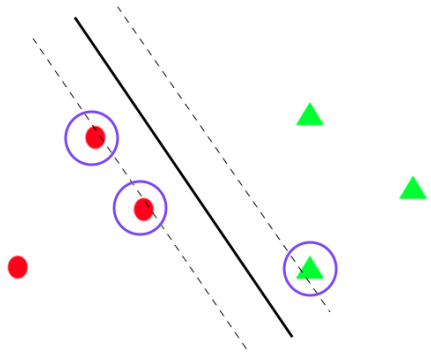
Question 2 Tune the depth of the tree

Tune the max_depth parameter of DecisionTreeClassifier. How does the training accuracy and test accuracy change when you vary the value of max_depth?

Increasing max_depth beyond 3 doesn't change the acccuracy on the training and test accuracy. Changing the max_depth to 2, decreased the training accuracy to 0.88, but increased the test accuracy to 0.8. Changing the max_depth to 1, decreased the training accuracy to 0.77, but the test accuracy stayed the same (compared to max_depth = 3) at 0.6.

3) SVM

a)



b)

Yes, changing the support vectors will change the margins thus changing the decision boundary. Changing the non support vectors points when they are not at boundary does not change the support vectors.

c) (next page)

Question 1 Implement SVM with sklearn

Implement SVM classifier to classify the dataset, and vary the value of C. What do you observe ?

I observe that with C = 1, the training data accuracy is 0.98. With C = 2 or above, the training is 1. The line on the graph splits the pos and neg points clearly.

```
from sklearn.svm import LinearSVC
################################################################################
# Scale the data and set up the SVM training                                   #
################################################################################

# scale the data

scaler = preprocessing.StandardScaler().fit(X)
scaleX = scaler.transform(X)

# add an intercept term and convert y values from [0,1] to [-1,1]

XX = np.array([(1,x1,x2) for (x1,x2) in scaleX])
yy = np.ones(y.shape)
yy[y == 0] = -1
yy[y == 0] = -1

################################################################################
#  Training linear SVM                                                         #
#  Train a linear SVM on the data set and the plot the learned                 #
#  decision boundary                                                           #
################################################################################

################################################################################
# TODO :: You will change this line below to vary C.                           #
################################################################################
C = 5

# TODO :: define your svm classifier by using sklearn LinearSVC; expect 1 line of code
svm = LinearSVC(C = C)
# TODO :: fit on your training data; expect 1 line of code
svm.fit(XX, yy)

# TODO :: classify the training data; expect 1 line of code
y_pred = svm.predict(XX)

print("Accuracy on training data = %.3f" %metrics.accuracy_score(yy,y_pred))

# visualize the decision boundarFy

utils.plot_decision_boundary(scaleX,y,svm,'x1','x2',['neg','pos'])
```

```
Accuracy on training data = 1.000
```

Question 2 Add PolynomialFeatures

Add polynomial features to the data and fit the LinearSVC with the new dataset. Tune the degree of the feature interaction to make the model correctly classify all the data in training set.

In [20]:
```
# TODO :: expect 13 - 15 lines of code

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=3)

X_transformed = poly.fit_transform(XX)

svm.fit(X_transformed, yy)

y_pred = svm.predict(X_transformed)
print("Accuracy on training data = %.3f" %metrics.accuracy_score(yy,y_pred))
```

```
Accuracy on training data = 1.000
```