Object Recognition and Image Understanding
Exercise Sheet 7

Sascha Stelling

June 12, 2018

Task 1 see knn.py
Code:

```
#!/usr/bin/env python2

from prepare_toy_data import prepare_toy_data
import random
import math
import operator
import numpy as np
from collections import Counter


def train(X_train, y_train):
# do nothing
return

def predict(X_train, y_train, x_test, k):
# create list for distances and targets
distances = []
targets = []

for i in range(len(X_train)):
subtr = x_test - X_train
distance = np.sqrt(np.sum(subtr))
distances.append([distance, i])

distances = sorted(distances)

# make list of the k neighbours' targets
for i in range(k):
index = distances[i][1]
targets.append(y_train[index])
```

```
return Counter(targets).most_common(1)[0][0]

def kNearestNeighbour(X_train, y_train, X_test, predictions, k):
# train on the input data
train(X_train, y_train)

for i in range(len(X_test)):
predictions.append(predict(X_train, y_train, X_test[i, :], k))

def main():
predictions = []
X_train, X_test, labels_train, labels_test = prepare_toy_data(300, False
k = 2
kNearestNeighbour(X_train, labels_train, X_test, predictions, k)

predictions = np.asarray(predictions)

accuracy = accuracy_score(labels_test, predictions)
print('Accuracy of this classifier: ' + accuracy*100 + '%')

main()
```

Task 2

1. $\mathcal{L}_{CE}(x) = -\log(p_y(x))$
   $p_y(x) = p(class = y|x) = \frac{e^{s_y(x)}}{\sum_{i=1}^{C} e^{s_i(x)}}$
   $s_i(x) = \omega_t^T x + b_i$
   $\frac{\partial s_i}{\partial \omega_i} = x$
   $\frac{\partial s_i}{\partial b_i} = 1$
   $\frac{\partial p_y}{\partial s_i} = p_i(1 - p_i), i = j$
   $\frac{\partial p_y}{\partial s_i} = -p_i p_j, i \neq j$
   $\frac{\partial \mathcal{L}_{CE}}{\partial p_y} = -\frac{1}{p_y}\frac{\partial p_y}{\partial s_i}$
   $\frac{\partial \mathcal{L}_{CE}}{\partial \omega_i} = -\frac{1}{p_y}\frac{\partial p_y}{\partial s_i} p_i(1 - p_i)x = (p_i - 1)x$ if $i = y$
   $\frac{\partial \mathcal{L}_{CE}}{\partial \omega_i} = -\frac{1}{p_y}\frac{\partial p_y}{\partial s_i} - p_i p_j x = p_i x$ if $i \neq y$
   $\frac{\partial \mathcal{L}_{CE}}{\partial b_i} = -\frac{1}{p_y}\frac{\partial p_y}{\partial s_i} p_i(1 - p_i) = (p_i - 1)$ if $i = y$
   $\frac{\partial \mathcal{L}_{CE}}{\partial b_i} = -\frac{1}{p_y}\frac{\partial p_y}{\partial s_i} - p_i p_j = p_i$ if $i \neq y$

2. See lin.py:

```
import numpy as np
```

```
from prepare_toy_data import prepare_toy_data
from pr import prcurve

class Perceptron(object):
def __init__(self, eta=0.01, n_iter=10):
        self.eta = eta
        self.n_iter = n_iter

def fit(self, X, y):

        self.w_ = np.zeros(1 + X.shape[1])
        self.errors_ = []


for _ in range(self.n_iter):
        errors = 0
                for xi, target in zip(X, y):
                        update = self.eta * (target - self.predict(xi))
                        self.w_[1:] += update * xi
                        self.w_[0] += update
                        errors += int(update != 0.0)
                self.errors_.append(errors)
        return self

def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

def predict(self, X):
        return np.where(self.net_input(X) >= 0.0, 1, 0)



X_train, X_test, labels_train, labels_test = prepare_toy_data(300, False
alpha = 0.5
iterations = 20
p = Perceptron(alpha, iterations)

p.fit(np.array(X_train), labels_train)

predicted_test = p.predict(X_test)

prcurve(predicted_test, labels_test)
```

3. See pr.py:

```
import numpy as np
```

```
import matplotlib.pyplot as plt

def prcurve(predicted_label, true_label):

TP = np.zeros(len(predicted_label))
FP = np.zeros(len(predicted_label))
FN = np.zeros(len(predicted_label))


for i in range(len(predicted_label)):
        if(predicted_label[i]==true_label[i]):
                TP[i]=1
        elif(predicted_label[i]==1 and true_label[i]==0):
                FP[i]=1
        elif(predicted_label[i]==0 and true_label[i]==1):
                FN[i]=1



recall=TP/(TP+FP)
precision=TP/(TP+FN)
precision2=precision.copy()
i=recall.shape[0]-2

# interpolation
while i>=0:
        if precision[i+1]>precision[i]:
                precision[i]=precision[i+1]
        i=i-1

# plotting
fig, ax = plt.subplots()
for i in range(recall.shape[0]-1):
ax.plot((recall[i],recall[i]),(precision[i],
 precision[i+1]),'k-',label='',color='red') #vertical
ax.plot((recall[i],recall[i+1]),(precision[i+1],
 precision[i+1]),'k-',label='',color='red') #horizontal

ax.plot(recall,precision2,'k--',color='blue')
#ax.legend()
ax.set_xlabel("recall")
ax.set_ylabel("precision")
plt.savefig('fig.jpg')
fig.show()
```