1. ВЫБОР ДАТАСЕТА

Для реализации домашней работы по анализу данных по курсу "Упорядоченные множества в анализе данных" был выбран датасет с результатами двух разных видов терапии, применяемых для лечения детей с онкологическими заболеваниями.

Количество объектов:
778

Краткое описание полей:
ID - Идентификационный номер пациента
Sex, Immun, CNS, Mediastinum, Zytogen, Region, Geb_month, Diag_month, syndrome - категориальные признаки
Age, Leuc, Leber, Milz, height, weight - количественные признаки
Better - какое из двух видов лечения лучше. Если 0, то нельзя сказать, какое лучше

2. ПЕРЕСМОТР ПРОСТРАНСТВА ПРИЗНАКОВ В ВЫБРАННОМ ДАТАСЕТЕ

- Импорт датасета

```
import pandas as pd

data = pd.read_csv('2008_200_300.csv')
```

- Исключение из анализа объектов с пустыми значениями по признакам

```
# исключение 24 объектов с пустыми значениями по признаку Zytogen
# и 6 объектов с пустыми значениями по признаку Region
data = data.dropna()

# исключение 6 объектов с нулевыми значениями по признаку Better
data = data.drop(data[data.Better == 0].index)
```

- Преобразование количественных признаков в категориальные

```
data['Age_c'] = pd.qcut(data['Age'], 5)
data['Leuc_c'] = pd.qcut(data['Leuc'], 5)
data['Leber_c'] = pd.cut(data['Leber'], 8)
data['Milz_c'] = pd.cut(data['Milz'], 5)
data['height_c'] = pd.qcut(data['height'], 5)
data['weight_c'] = pd.qcut(data['weight'], 5)

# Удаление столбцов с количественными признаками + удаление столбца ID
data = data.drop(['ID','Age','Leuc','Leber','Milz','height','weight'], 1)
```

- Преобразование категориальных признаков в бинарные

```
data = pd.get_dummies(data, columns = ['Sex', 'Immun', 'CNS', 'Mediastinum',
'Zytogen', 'Region', 'Geb_month', 'Diag_month', 'syndrome', 'Age_c', 'Leuc_c',
'Leber_c', 'Milz_c', 'height_c', 'weight_c'])
```

- Замена терапии "300" на "positive", терапии "200" на "negative" для удобства последующего анализа

```
data.loc[data.Better == 300, 'Better'] = 'positive'
data.loc[data.Better == 200, 'Better'] = 'negative'
```

- Сохранение нового датасета

```
data.to_csv('2008_200_300_new.csv',index=False)
```

3. КРОСС-ВАЛИДАЦИЯ

```
newdata = pd.read_csv('2008_200_300_new.csv')

from sklearn.cross_validation import KFold

kf = KFold(len(newdata), n_folds=10, shuffle=True, random_state=None)
for k, (train, test) in enumerate(kf):
    newdata.iloc[train].to_csv('train'+str(k+1)+'.csv',index=False,header=False)
    newdata.iloc[test].to_csv('test'+str(k+1)+'.csv',index=False,header=False)
```

4. РЕАЛИЗАЦИЯ ВСПОМОГАТЕЛЬНЫХ ФУНКЦИЙ

- Загрузка train с разделением на положительный и отрицательный контексты + загрузка test

```
def load(i):
    q = open('train'+str(i)+'.csv', 'r')
    train = [a.strip().split(',') for a in q]
    plus = [a for a in train if a[0] == 'positive']
    minus = [a for a in train if a[0] == 'negative']
    q.close()

    w = open('test'+str(i)+'.csv','r')
    unknown = [a.strip().split(',') for a in w]
    w.close
    return plus, minus, unknown
```

- Приведение в необходимый формат рассматриваемого понятия

```
attrib_names = list(newdata)

def make_intent(example):
    return set([i+':'+str(k) for i, k in zip(attrib_names,example)])
```

- Создание формул для расчета метрик качества

```
def accuracy(r):
    return float(r["TP"] + r["TN"]) / max(1, r["TP"] + r["TN"] + r["FP"] + r["FN"]
+ r["contradictory"])

def precision(r):
    return float(r["TP"]) / max(1, r["TP"] + r["FP"])

def recall(r):
    return float(r["TP"]) / max(1, r["TP"] + r["FN"])

def results(r):
    metrics = {}
    metrics["accuracy"] = accuracy(r)
    metrics["precision"] = precision(r)
    metrics["recall"] = recall(r)
    return metrics
```

5. РЕАЛИЗАЦИЯ АЛГОРИТМОВ

Алгоритм 1

Пример классифицируется положительно, если каждое его пересечение с объектами из плюс-контекста вкладывается не более, чем в 2 описания из минус-контекста (и наоборот).

```
def algorithm1(context_plus, context_minus, example):
    eintent = make_intent(example)
    eintent.discard('Better:positive')
    eintent.discard('Better:negative')
    labels = {"positive": True, "negative": True}
    for e in context_plus:
        ei = make_intent(e)
        candidate_intent = ei & eintent
        closure = [make_intent(i) for i in context_minus if
make_intent(i).issuperset(candidate_intent)]
        if len(closure) > 2:
            labels["positive"] = False
            break
    for e in context_minus:
        ei = make_intent(e)
        candidate_intent = ei & eintent
```

```
        closure = [make_intent(i) for i in context_plus if
make_intent(i).issuperset(candidate_intent)]
        if len(closure) > 2:
            labels["negative"] = False
            break

    if not labels["positive"] ^ labels["negative"]:
       return "contradictory"
    if example[0] == "positive" and labels["positive"]:
       return "TP"
    if example[0] == "positive" and labels["negative"]:
       return "FN"
    if example[0] == "negative" and labels["positive"]:
       return "FP"
    if example[0] == "negative" and labels["negative"]:
       return "TN"
```

Алгоритм 2

Каждый объект из плюс-контекста 'голосует' за положительную классификацию, если его пересечение с примером не вкладывается в описания из минус-конекста (и наоборот). Пример классифицируется положительно, если преобладает количество 'голосов' за положительную классификацию (и наоборот).

```
def algorithm2(context_plus, context_minus, example):
    eintent = make_intent(example)
    eintent.discard('Better:positive')
    eintent.discard('Better:negative')
    labels = {"positive": 0, "negative": 0}
    for e in context_plus:
        ei = make_intent(e)
        candidate_intent = ei & eintent
        closure = [make_intent(i) for i in context_minus if
make_intent(i).issuperset(candidate_intent)]
        if not closure:
            labels["positive"] += 1
    for e in context_minus:
        ei = make_intent(e)
        candidate_intent = ei & eintent
        closure = [make_intent(i) for i in context_plus if
make_intent(i).issuperset(candidate_intent)]
        if not closure:
            labels["negative"] += 1

    labels["positive"] = float(labels["positive"]) / len(context_plus)
    labels["negative"] = float(labels["negative"]) / len(context_minus)
    if labels["positive"] > labels["negative"]:
        if example[0] == "positive":
            return "TP"
        else:
            return "FP"
    elif labels["positive"] < labels["negative"]:
        if example[0] == "negative":
```

```
            return "TN"
        else:
            return "FN"
    else:
       return "contradictory"
```

Алгоритм 3

Пример классифицируется положительно, если его относительная поддержка в плюс-контексте превышает его относительную поддержку в минус-контексте (и наоборот).

```
def algorithm3(context_plus, context_minus, example):
    eintent = make_intent(example)
    eintent.discard('Better:positive')
    eintent.discard('Better:negative')
    labels = {"positive": 0, "negative": 0}
    len_context_plus = len(context_plus)
    len_context_minus = len(context_minus)
    for e in context_plus:
        ei = make_intent(e)
        candidate_intent = ei & eintent
        support = [make_intent(i) for i in context_plus if
make_intent(i).issuperset(candidate_intent)]
        labels["positive"] += float(len(support)) / len_context_plus
    for e in context_minus:
        ei = make_intent(e)
        candidate_intent = ei & eintent
        support = [make_intent(i) for i in context_minus if
make_intent(i).issuperset(candidate_intent)]
        labels["negative"] += float(len(support)) / len_context_minus

    labels["positive"] = float(labels["positive"]) / len_context_plus
    labels["negative"] = float(labels["negative"]) / len_context_minus
    if labels["positive"] > labels["negative"]:
        if example[0] == "positive":
            return "TP"
        else:
            return "FP"
    elif labels["positive"] < labels["negative"]:
        if example[0] == "negative":
            return "TN"
        else:
            return "FN"
    else:
       return "contradictory"
```

Алгоритм 4

Продолжая тему наивности (ха-ха), давайте реализуем алгоритм Наивного Байеса. Он является одним из самых известных алгоритмов машинного обучения, основной задачей

которого является восстановление плотностей распределения данных обучающей выборки.

```
def BernoulliNaiveBayes():
    # time on
    import timeit
    start = timeit.default_timer()

    # open binary dataset and separate the data from the target attributes
    q = open('2008_200_300_new.csv', 'r')
    dataset = [a.strip().split(',') for a in q]
    dataset = dataset[1:]
    q.close()

    import numpy as np
    A = np.array(dataset)
    X = A[:,1:].astype(float)
    Y = A[:,0]

    # run BernoulliNB() on our dataset
    from sklearn.naive_bayes import BernoulliNB
    from sklearn import cross_validation
    from sklearn.preprocessing import LabelBinarizer
    model = BernoulliNB()
    model.fit(X, Y)

    # find out results
    acc = np.mean(cross_validation.cross_val_score(model, X, Y, cv=10))

    # binarize target to find precision and recall
    lb = LabelBinarizer()
    Y = np.array([number[0] for number in lb.fit_transform(Y)])

    prec = np.mean(cross_validation.cross_val_score(model, X, Y, cv=10, scoring =
'precision'))
    rec = np.mean(cross_validation.cross_val_score(model, X, Y, cv=10, scoring =
'recall'))

    # time off
    stop = timeit.default_timer()
    time = stop - start

    return acc, prec, rec, time
```

6. ЗАПУСК АЛГОРИТМОВ И ПОДВЕДЕНИЕ ИТОГОВ ПО МЕТРИКАМ

- Функция запуска алгоритмов (для 1-3) и подведения итогов

```
def summary(algorithm_name):
    # time on
    import timeit
    start = timeit.default_timer()
```

```
    acc = 0
    prec = 0
    rec = 0
    for index in range(1,11):
        (iplus, iminus, iunknown) = load(index)
        cv_res = {
            "TP": 0,
            "TN": 0,
            "FP": 0,
            "FN": 0,
            "contradictory": 0,
            }
        for elem in iunknown:
            pin = algorithm_name(iplus, iminus, elem)
            cv_res[pin] += 1
        res = results(cv_res)
        acc += res['accuracy']
        prec += res['precision']
        rec += res['recall']

    # find mean results for cross-validation
    acc = acc/10
    prec = prec/10
    rec = rec/10

    # time off
    stop = timeit.default_timer()
    time = stop - start

    return acc, prec, rec, time
```

- Запуск Алгоритма 1 и подведение итогов

```
(a1,p1,r1,time1) = summary(algorithm1)
print('Accurcy: '+str(a1*100)+'%')
print('Precision: '+str(p1*100)+'%')
print('Recall: '+str(r1*100)+'%')
print('Time of algorithm work: '+str(time1))
```

```
Accurcy: 12.772972972972973%
Precision: 96.73076923076923%
Recall: 95.32467532467534%
Time of algorithm work: 9304.81378628299
```

- Запуск Алгоритма 2 и подведение итогов

```
(a2,p2,r2,time2) = summary(algorithm2)
print('Accurcy: '+str(a2*100)+'%')
print('Precision: '+str(p2*100)+'%')
print('Recall: '+str(r2*100)+'%')
print('Time of algorithm work: '+str(time2))
```

```
Accurcy: 77.01981981981983%
Precision: 80.6285098120081%
Recall: 86.90841576918024%
Time of algorithm work: 8877.838668775003
```

- Запуск Алгоритма 3 и подведение итогов

```
(a3,p3,r3,time3) = summary(algorithm3)
print('Accurcy: '+str(a3*100)+'%')
print('Precision: '+str(p3*100)+'%')
print('Recall: '+str(r3*100)+'%')
print('Time of algorithm work: '+str(time3))
```

```
Accurcy: 33.59639639639639%
Precision: 20.0%
Recall: 0.5744520030234316%
Time of algorithm work: 11359.37753788702
```

- Запуск Алгоритма 4 и подведение итогов

```
(a4,p4,r4,time4) = BernoulliNaiveBayes()
print('Accurcy: '+str(a4*100)+'%')
print('Precision: '+str(p4*100)+'%')
print('Recall: '+str(r4*100)+'%')
print('Time of algorithm work: '+str(time4))
```

```
Accurcy: 80.3397260274%
Precision: 89.6280133232%
Recall: 79.8489795918%
Time of algorithm work: 0.17655974900117144
```

Таким образом, наиболее высокую точность из представленных выше "ленивых" алгоритмов дал Алгоритм 2 (с 'голосованием'). Его точность почти совпадает с точностью алгоритма Наивного Байеса, однако время работы алгоритма сильно уступает.