Алёна Егорова, 494, задание 1

Здесь будет много комментариев, создавала я их, чтобы лучше понять и потом если что посмотереть, как и что я делала

In [29]:

```
from matplotlib.colors import ListedColormap
from sklearn import cross_validation, datasets, metrics, neighbors
from matplotlib import pyplot
import numpy as np
```

Генерируем выборку размера 1000 с 2 признаками и 4 классами

In [69]:

```
sample = datasets.make_classification(n_samples=100, n_features=2,
                                      n_informative=2,
                                      n_redundant=0, n_classes=4,
                                      n_clusters_per_class=1,
                                      shuffle=True, random_state=1)
```

Визуализирую разделяющие поверхности для k = 1, 2, 3, 4, 5

In [78]:

```
colors = ListedColormap(['red', 'blue', 'yellow', 'green']) #  классы
light_colors = ListedColormap(['lightcoral', 'lightblue',
                               'lightyellow', 'lightgreen'])

def get_meshgrid(data, step=.05, border=.5,):
    x_min, x_max = data[:, 0].min() - border, data[:, 0].max() + border
    y_min, y_max = data[:, 1].min() - border, data[:, 1].max() + border
    return np.meshgrid(np.arange(x_min, x_max, step),
                       np.arange(y_min, y_max, step))

def plot_decision_surface(estimator, train_data, train_labels,
                          test_data, test_labels,
                          colors = colors, light_colors = light_colors):
    #fit model
    estimator.fit(train_data, train_labels)

    #set figure size
    pyplot.figure(figsize = (16, 6))

    #plot decision surface on the train data
    pyplot.subplot(1,2,1)
    xx, yy = get_meshgrid(train_data)
    mesh_predictions = np.array(estimator.predict(np.c_[xx.ravel(),
                                        yy.ravel()])).reshape(xx.shape)
    pyplot.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
    pyplot.scatter(train_data[:, 0], train_data[:, 1], c = train_labels,
                   s = 100, cmap = colors)
    pyplot.title('Train data, accuracy={:.2f}'.format(metrics.accuracy_score(
        train_labels, estimator.predict(train_data))))

    #plot decision surface on the test data
    pyplot.subplot(1,2,2)
    pyplot.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
    pyplot.scatter(test_data[:, 0], test_data[:, 1], c = test_labels,
                   s = 100, cmap = colors)
    pyplot.title('Test data, accuracy={:.2f}'.format(metrics.accuracy_score(
        test_labels,estimator.predict(test_data))))
    pyplot.show()
```

Разделили данные на кросс-валидации: обучающая выборка составляет 0.3 от всей

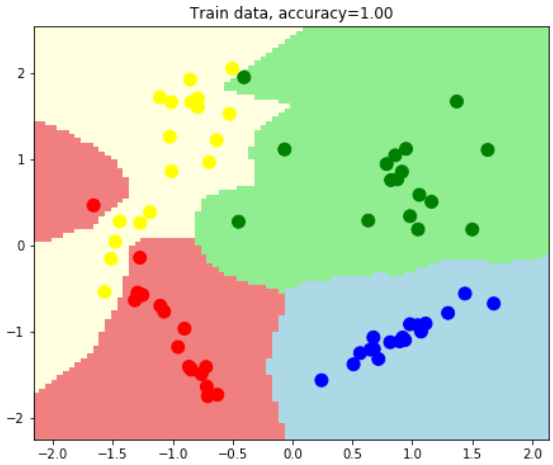
In [90]:

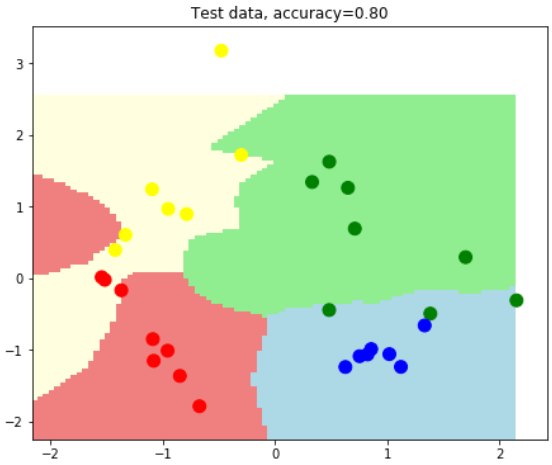
```
train_data, test_data, train_labels, test_labels = cross_validation.train_test_spli
                                        sample[0], sample[1],
                                        test_size = 0.3, random_state = 1)
```

In [92]:

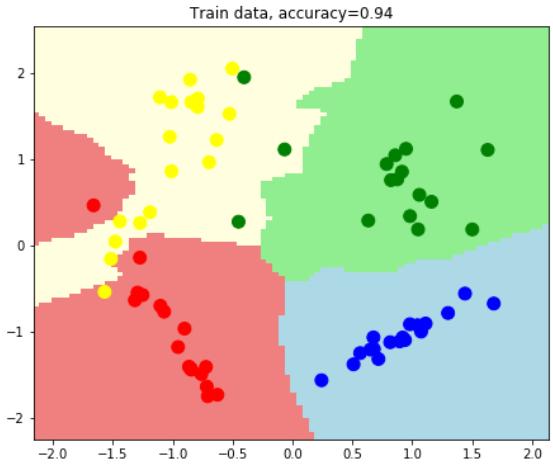
```
for i in range(1, 6):
    print("AMOUNT OF NEIGHBORS = " + str(i))
    estimator = neighbors.KNeighborsClassifier(n_neighbors=i)
    plot_decision_surface(estimator, train_data, train_labels,
                          test_data, test_labels)
```

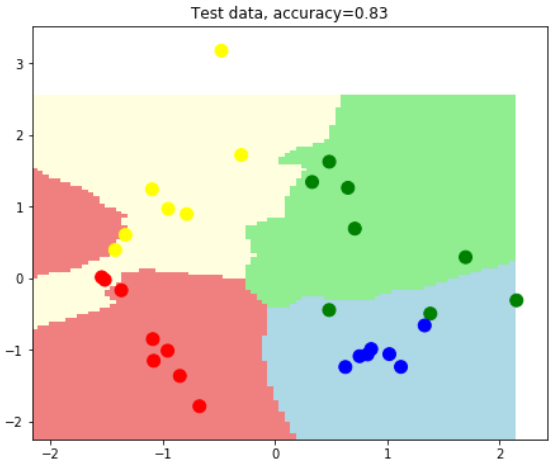
AMOUNT OF NEIGHBORS = 1



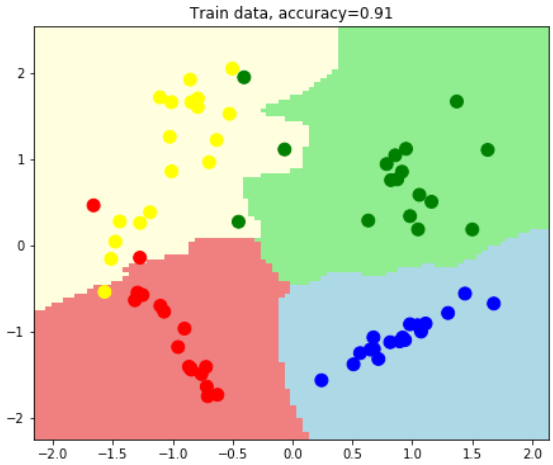


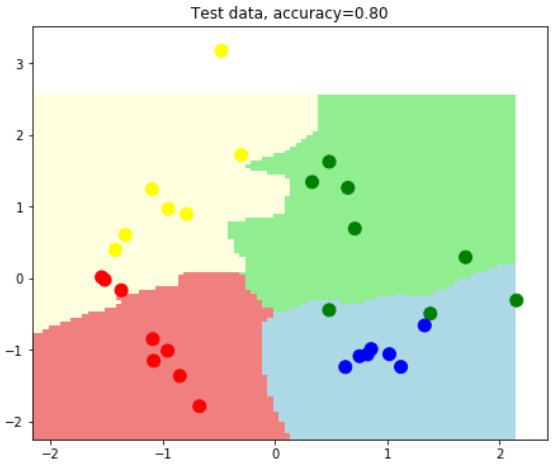
AMOUNT OF NEIGHBORS = 2



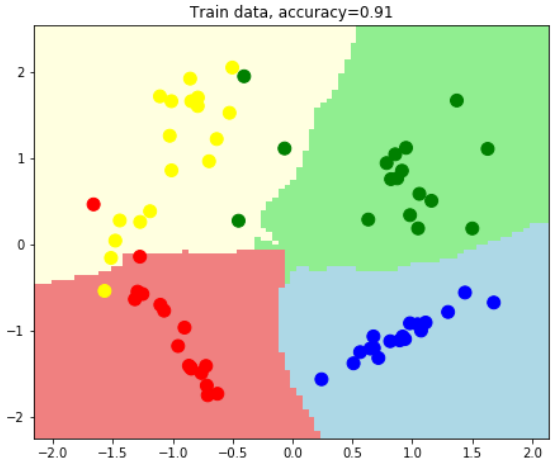


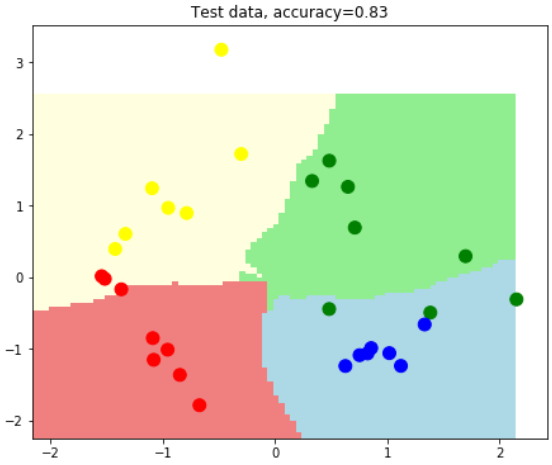
AMOUNT OF NEIGHBORS = 3



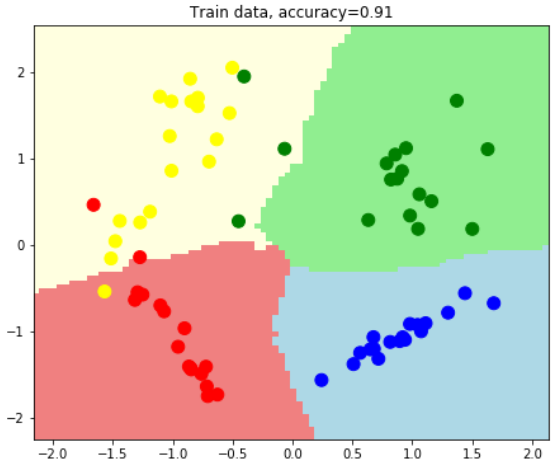


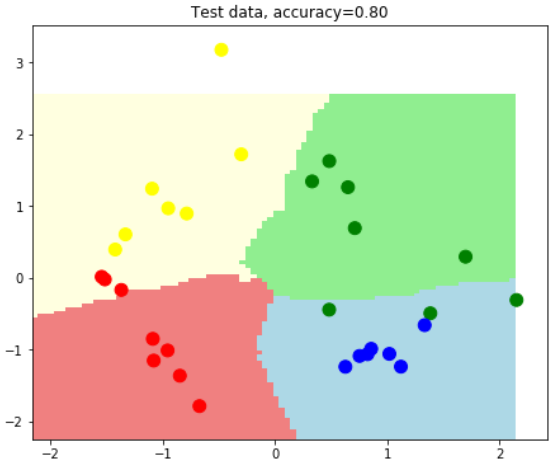
AMOUNT OF NEIGHBORS = 4





AMOUNT OF NEIGHBORS = 5





Попробуем подобрать оптимальное k с помощью 5-fold cross-validation

С помощью функции cross_val_score получим accuracy для каждого тестирования выборки на наших данных, разделенных на 5 fold'ов. Далее усредним accuracy для каждого значения количества соседей k. Нарисуем график (accuracy, k), найдем оптимальный k (k = 1, ..., 50. Взяла большое k, чтобы было нагляднее).

In [93]:

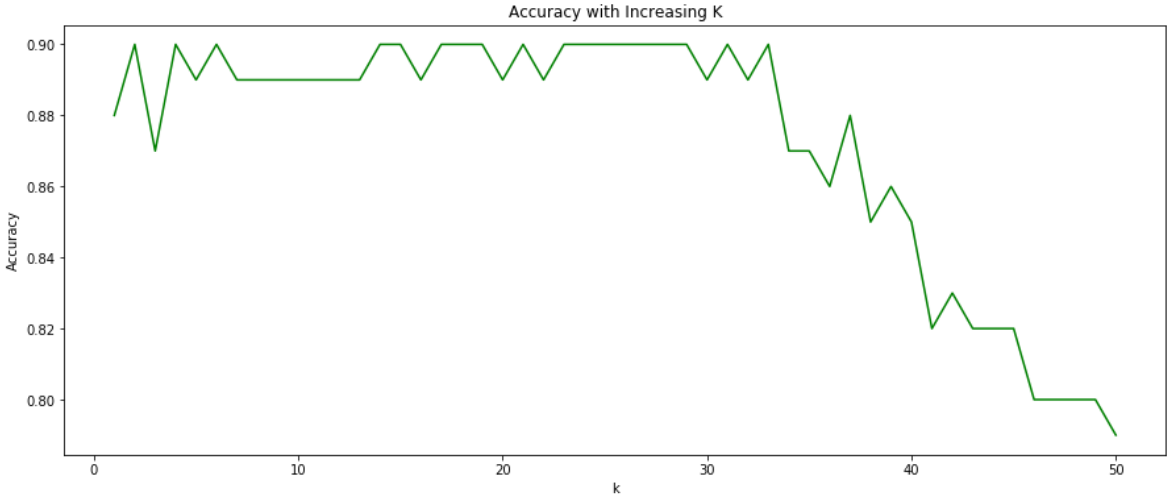
```
from sklearn.cross_validation import cross_val_score

accuracy = []
k = range(1, 51)

for i in range(1, 51):
    knn = neighbors.KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score(knn, sample[0], sample[1],
                             cv=5, scoring='accuracy')
    accuracy.append(scores.mean())
```

In [94]:

```
pyplot.figure(figsize=(15, 6))
pyplot.plot(k, accuracy, c='g')
#pyplot.scatter(k, accuracy, c='r')
pyplot.title("Accuracy with Increasing K")
pyplot.xlabel('k')
pyplot.ylabel('Accuracy')
pyplot.show()
```



Какие можно сделать выводы? Во-первых, хорошая точность показывается, когда мы рассматриваем четное число соседей (k - четно). А также, что после k = 7, точно стабилизировалась, значит алгоритм чрезмерно устойчив и такое значение k лучше не рассматривать.

Поэтому оптимальное значение в моём случае, наверное, k = 6.

P.S. Пробовала рисовать разделяющие поверхности при sample_size > 100, но там совсем ничего не видно.