Linear models for classification

3

Zukhra Abdiakhmetova

In this lecture

Binary classification

Multiclass classification

Naive Bayes classifiers

Decision trees

Binary classification

- For prediction use next expression

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \ldots + w[p] * x[p] + b > 0$$

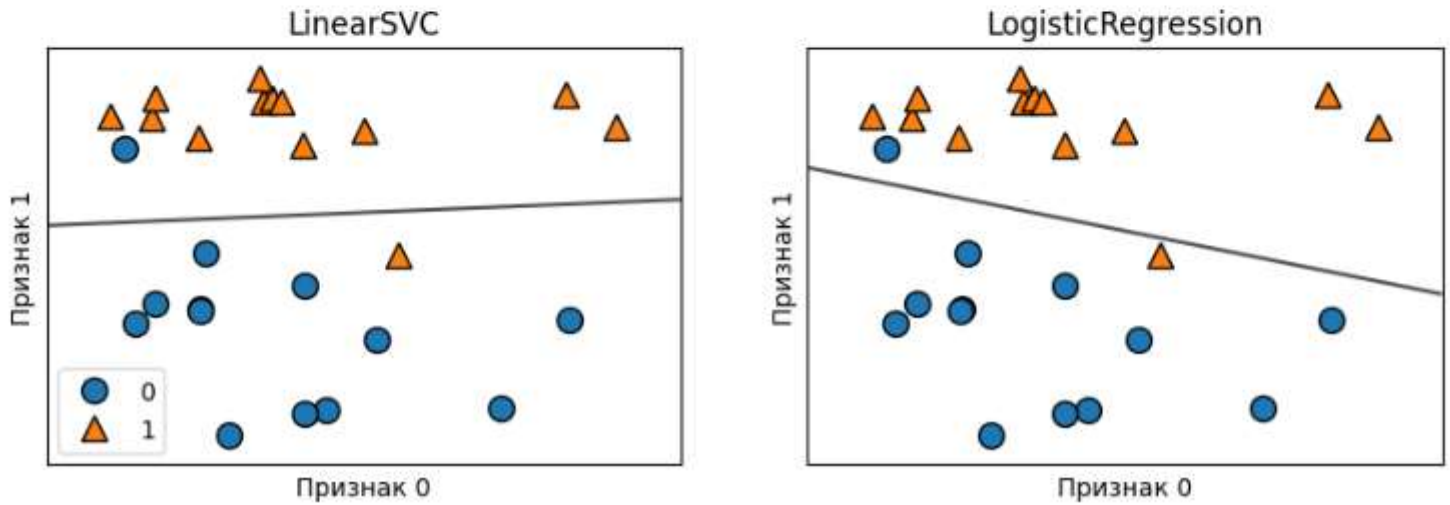
Linear models for classification:

- **logistic regression** - linear_model.LogisticRegression
- **linear support vector machines** - svm.LinearSVC

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
X, y = mglearn.datasets.make_forge()
fig, axes = plt.subplots(1, 2, figsize=(10, 3))

for model, ax in zip([LinearSVC(), LogisticRegression()], axes):
 clf = model.fit(X, y)
 mglearn.plots.plot_2d_separator(clf, X, fill=False, eps=0.5,
 ax=ax, alpha=.7)
 mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
 ax.set_title("{}".format(clf.__class__.__name__))
 ax.set_xlabel("Признак 0")
 ax.set_ylabel("Признак 1")
axes[0].legend()
```

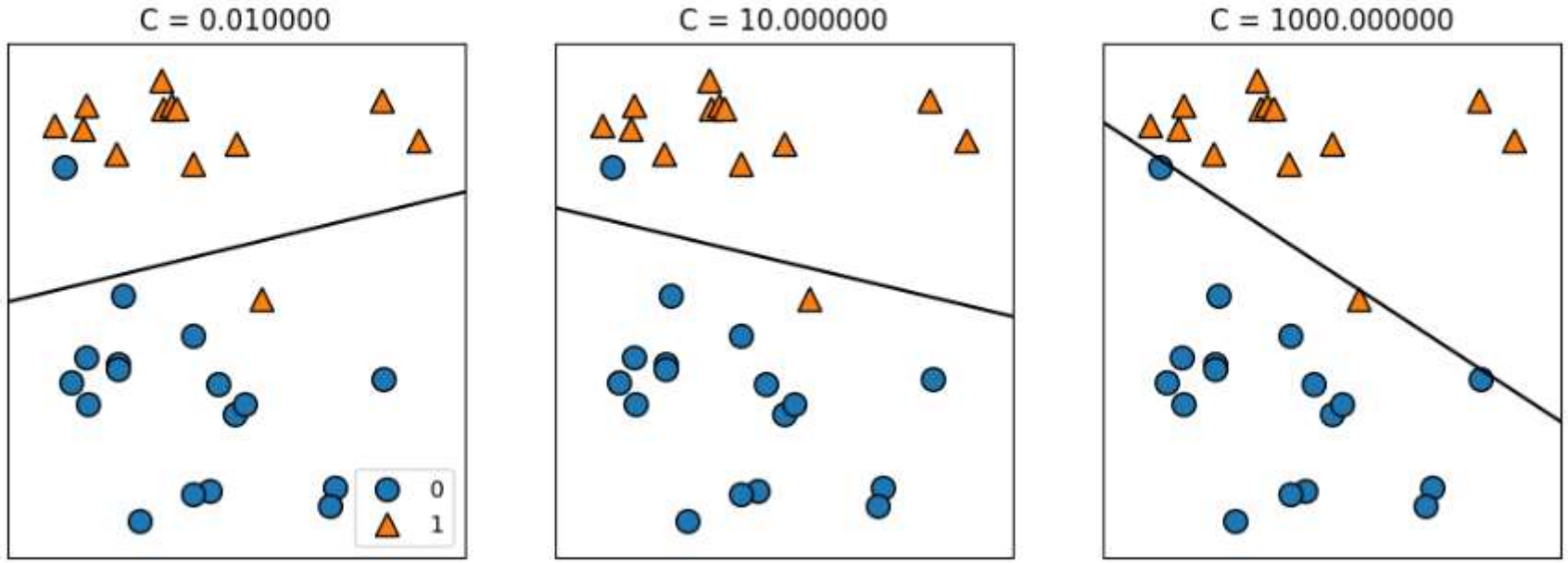
```
C:\Users\abdyahmetova.zuhra2\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\svm\_classes.py:31: Futu
ual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\abdyahmetova.zuhra2\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\svm\_base.py:1237: Conve
converge, increase the number of iterations.
  warnings.warn(
```

```
<matplotlib.legend.Legend at 0x27e871e4380>
```



mglearn.plots.plot_linear_svc_regularization()

```
mglearn.plots.plot_linear_svc_regularization()
```



```python
from sklearn.model_selection import train_test_split
```

```python
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
 cancer.data, cancer.target, stratify=cancer.target, random_state=42)
logreg = LogisticRegression().fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(logreg.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(logreg.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 0.955
Правильность на тестовом наборе: 0.951
```

C=1
undertraining

```python
logreg100 = LogisticRegression(C=100).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(logreg100.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(logreg100.score(X_test, y_test)))
```

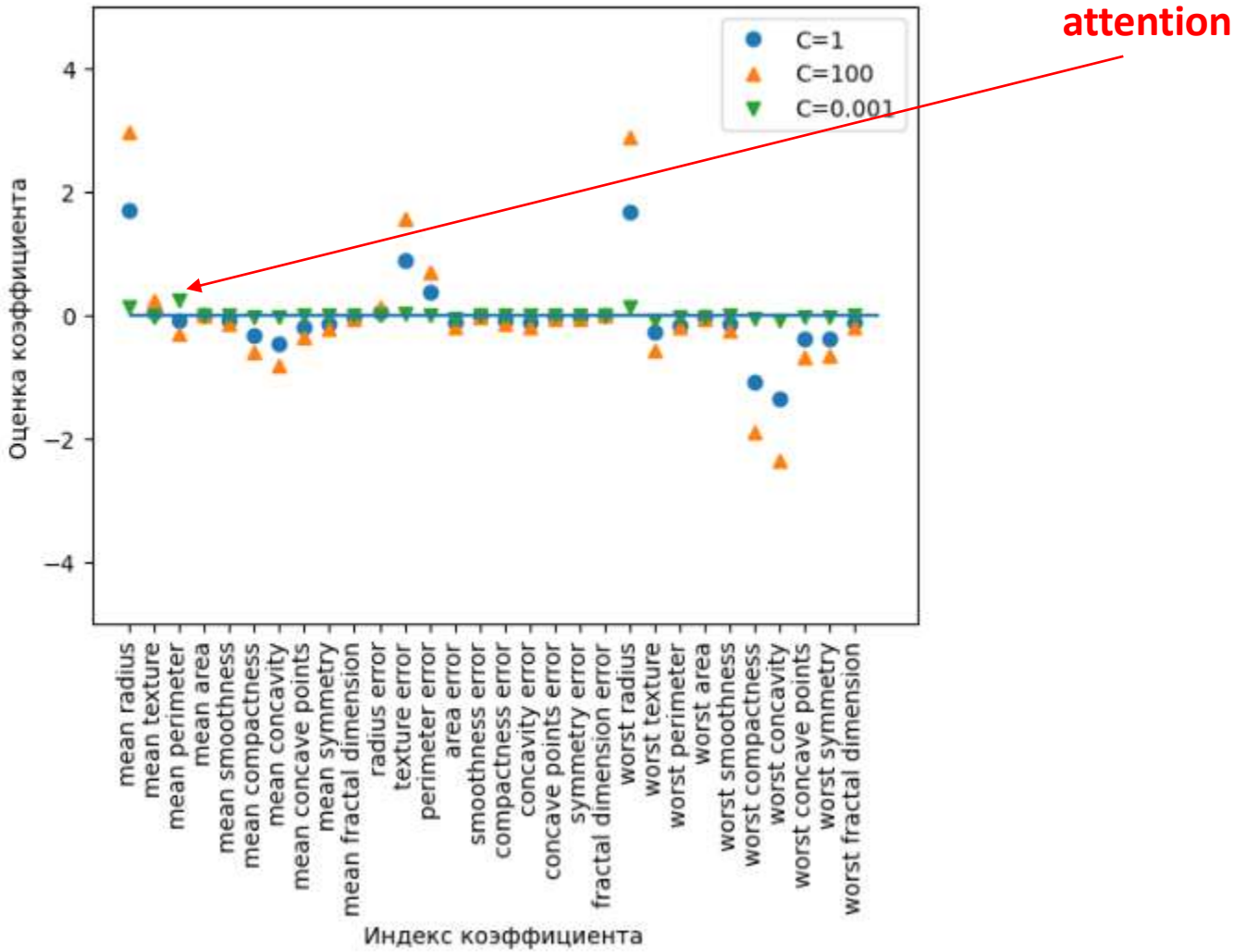
```
Правильность на обучающем наборе: 0.955
Правильность на тестовом наборе: 0.965
```

C=100

```python
logreg001 = LogisticRegression(C=0.01).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(logreg001.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(logreg001.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 0.937
Правильность на тестовом наборе: 0.930
```

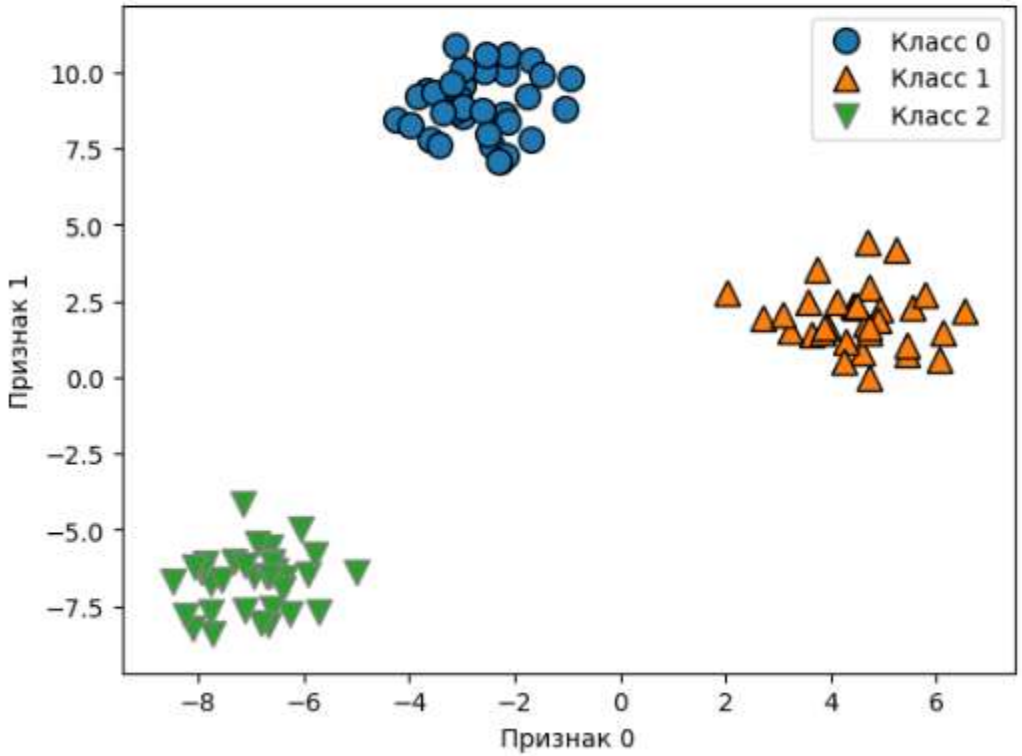
C=0,01
undertrainir


attention
C=1
C=100
C=0.001
Оценка коэффициента
4
2
0
−2
−4
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension
Индекс коэффициента

linear models for multiclass classification

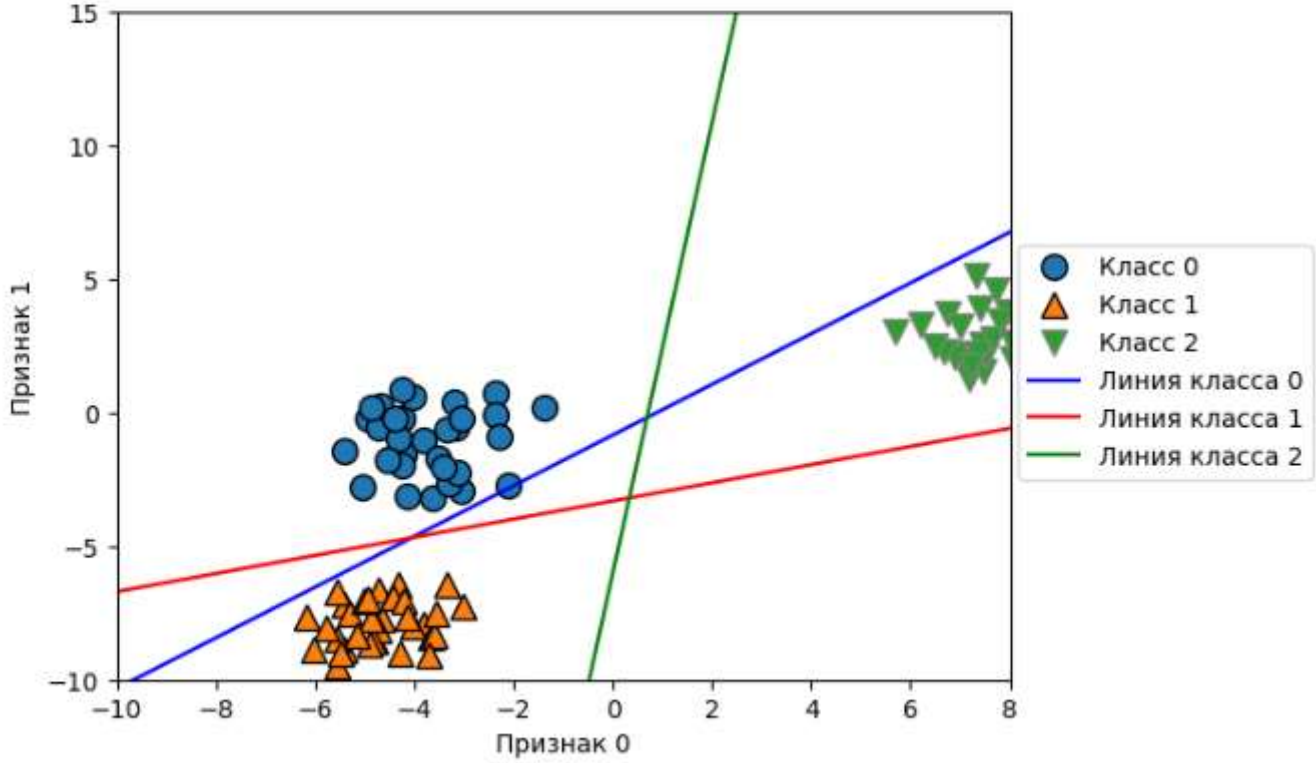
```
[26]: from sklearn.datasets import make_blobs
      X, y = make_blobs(random_state=42)
      mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
      plt.xlabel("Признак 0")
      plt.ylabel("Признак 1")
      plt.legend(["Класс 0", "Класс 1", "Класс 2"])
```

```
[26]: <matplotlib.legend.Legend at 0x27e894a2c90>
```



```
[32]: mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
 ['b', 'r', 'g']):
 plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.ylim(-10, 15)
plt.xlim(-10, 8)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.legend(['Класс 0', 'Класс 1', 'Класс 2', 'Линия класса 0', 'Линия класса 1',
 'Линия класса 2'], loc=(1.01, 0.3))
```

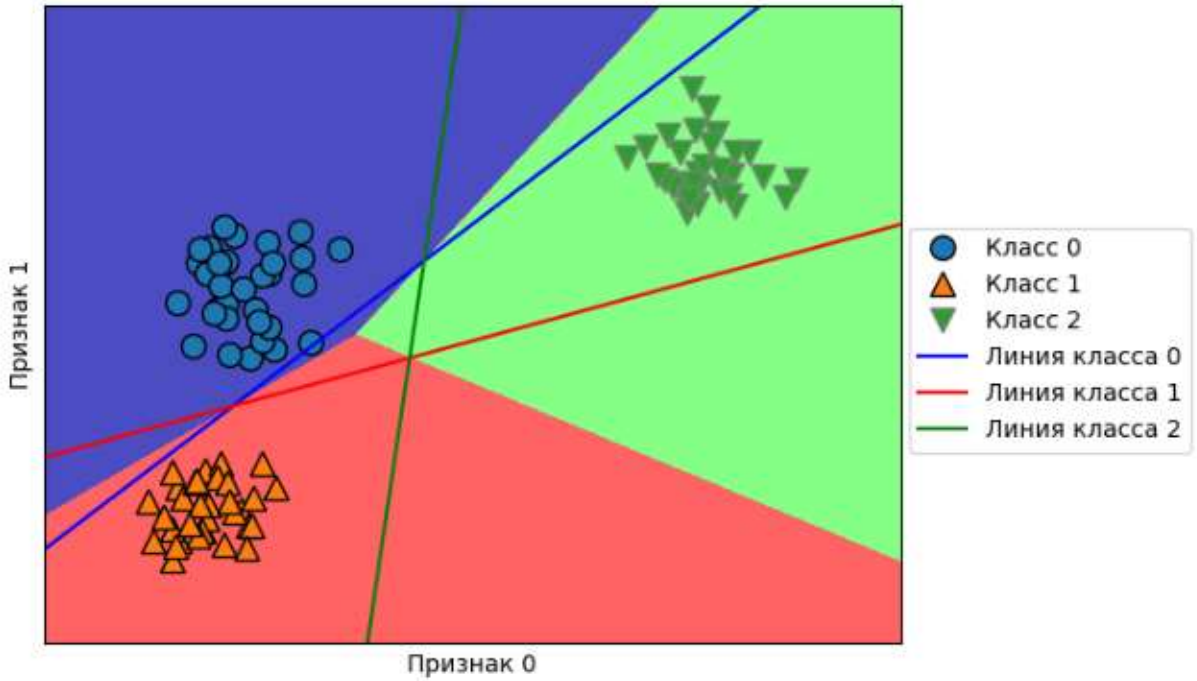
```
[32]: <matplotlib.legend.Legend at 0x27e8e9ca3c0>
```



For independent analyzing

```
[33]: mglearn.plots.plot_2d_classification(linear_svm, X, fill=True, alpha=.7)
      mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
      line = np.linspace(-15, 15)
      for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
       ['b', 'r', 'g']):
       plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
      plt.legend(['Класс 0', 'Класс 1', 'Класс 2', 'Линия класса 0', 'Линия класса 1',
       'Линия класса 2'], loc=(1.01, 0.3))
      plt.xlabel("Признак 0")
      plt.ylabel("Признак 1")
```

```
[33]: Text(0, 0.5, 'Признак 1')
```



pros and cons of linear models

- Big alpha and small C – simple models
- Linear models – fast in training and in prediction
- For data of handred thousand and mln lines recommended solver='sag' in LogisticRegression and Ridge
- SGDClassifier and SGDRegressor – classes, scalable versions of the linear models
- understandable

Naive Bayes classifiers

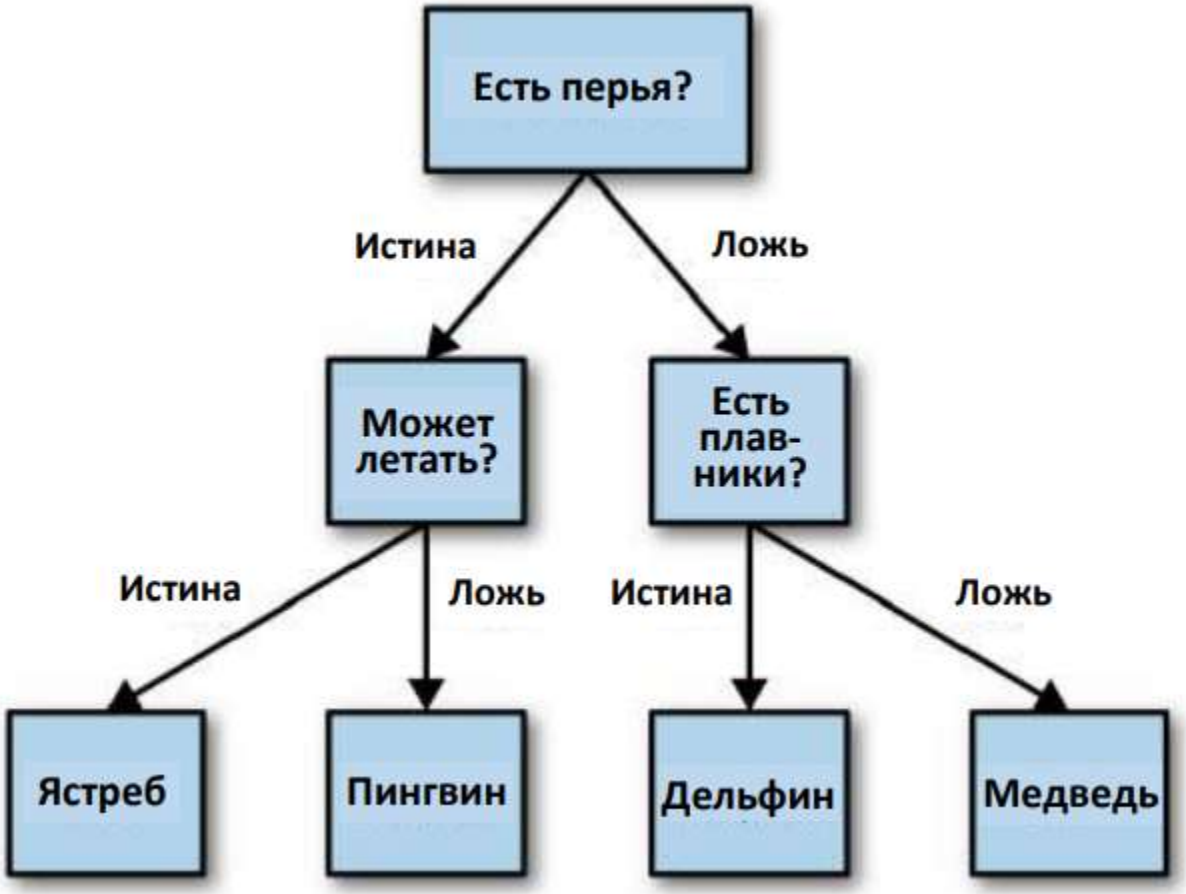
- Consider each feature separately

scikit-learn:

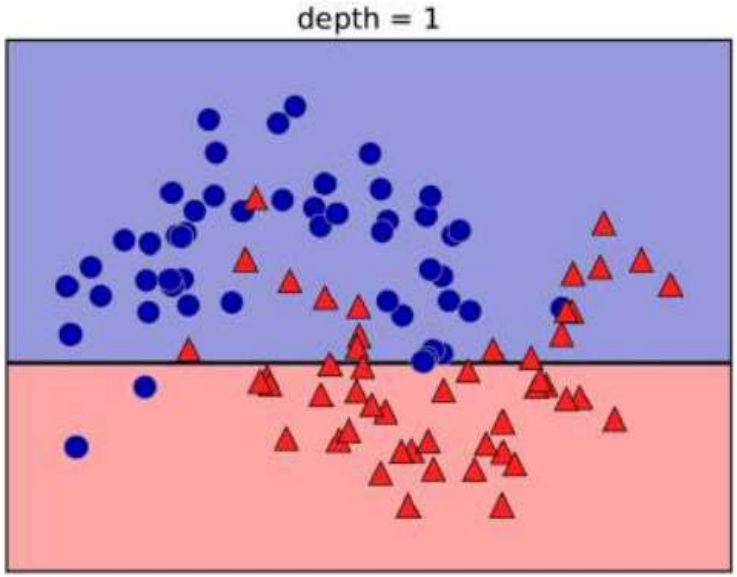
- GaussianNB – for all continuous data
- BernoulliNB – binary data , text data classification
- MultinomialNB – even or discrete data, text data classification

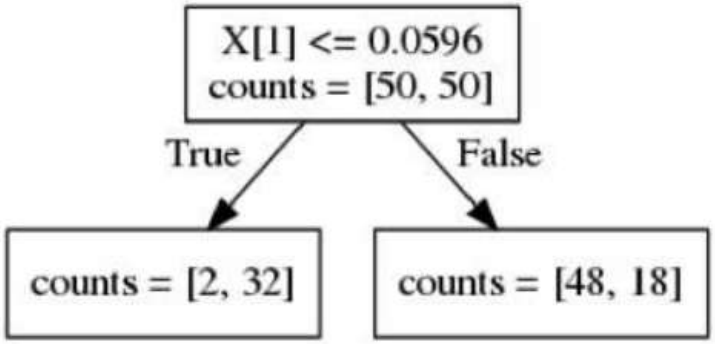
Decision trees

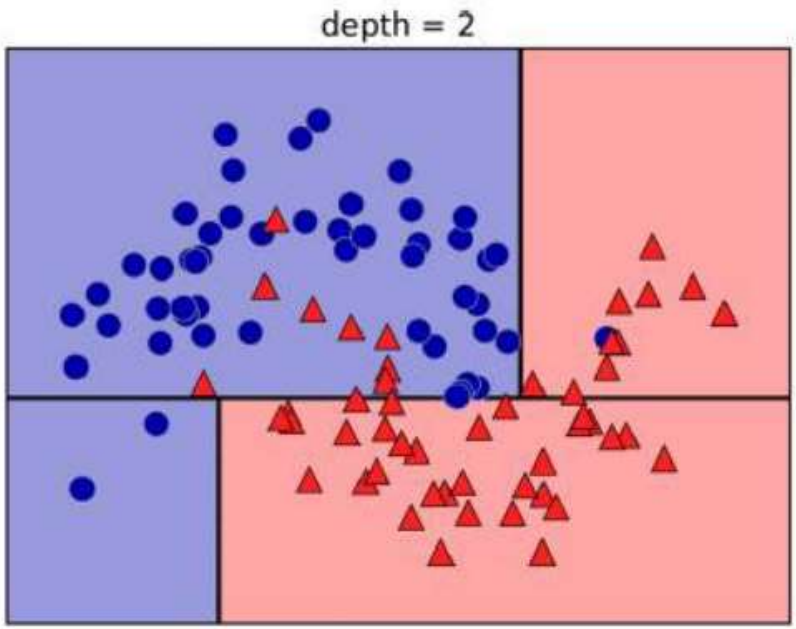
- For classification and regression problems

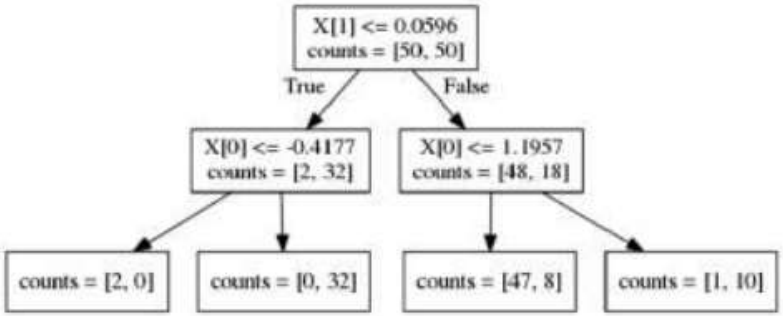


building a decision tree



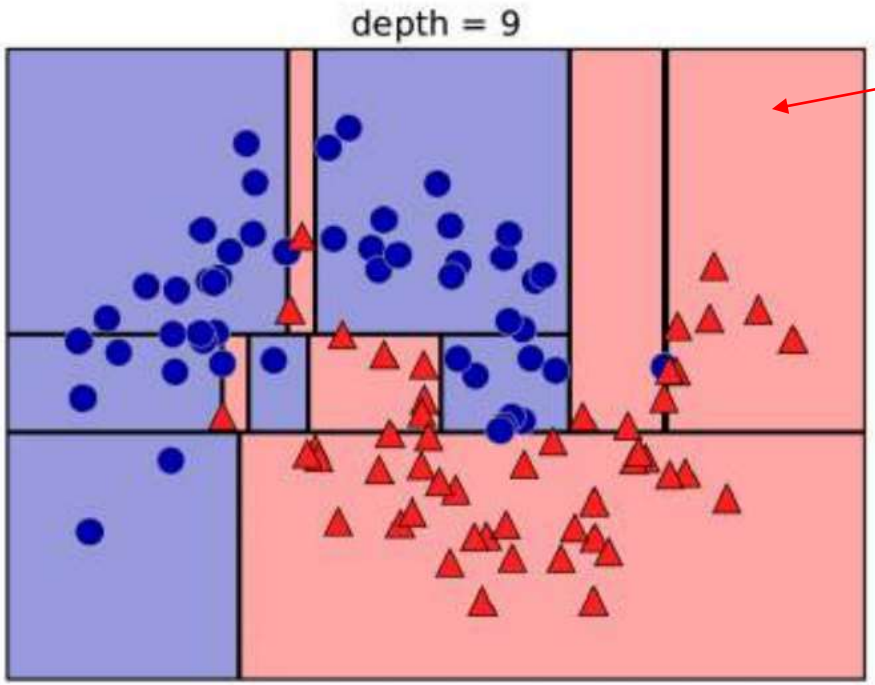


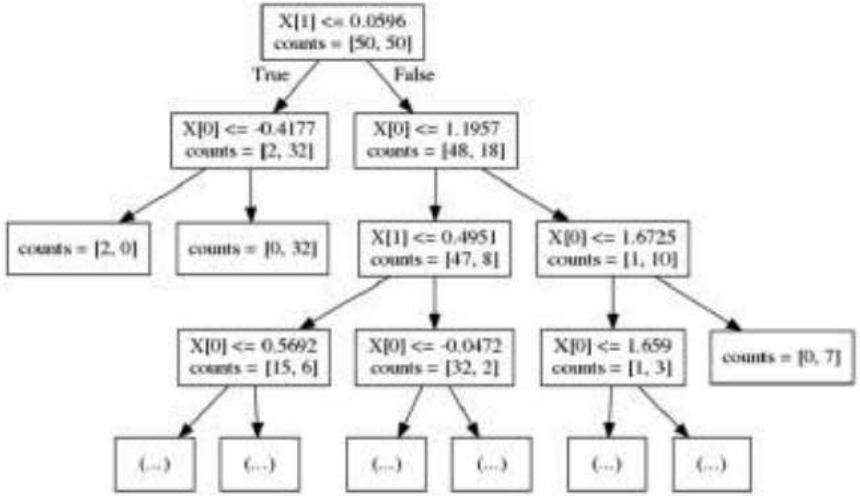




mglearn.plots.plot_tree_progressive()

Overtraining





two common strategies that allow prevent overtraining

- early stopping of tree construction, called pre-pruning
- building a tree with subsequent removal or reduction of uninformative nodes, called post-pruning or simply pruning
- **DecisionTreeRegressor** and **DecisionTreeClassifier**

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeClassifier
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
 cancer.data, cancer.target, stratify=cancer.target, random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(tree.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 1.000
Правильность на тестовом наборе: 0.937
```

```
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)
```

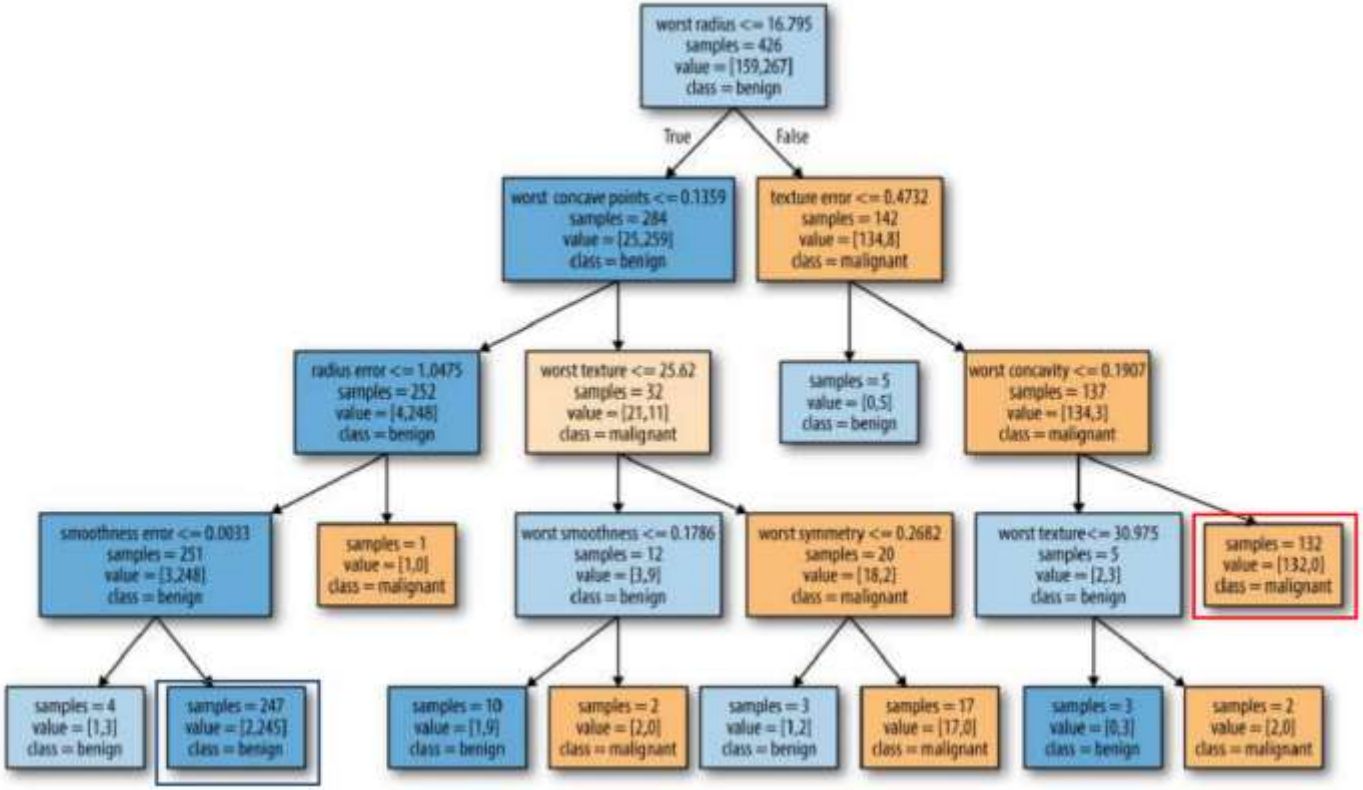
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, random_state=0)
```

```
print("Правильность на обучающем наборе: {:.3f}".format(tree.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 0.988
Правильность на тестовом наборе: 0.951
```

Visualization

```python
from sklearn.tree import export_graphviz
export_graphviz(tree, out_file="tree.dot", class_names=["malignant", "benign"],feature_names=cancer.feature_names, impurity=False, filled=True)

import graphviz
with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```



feature importance

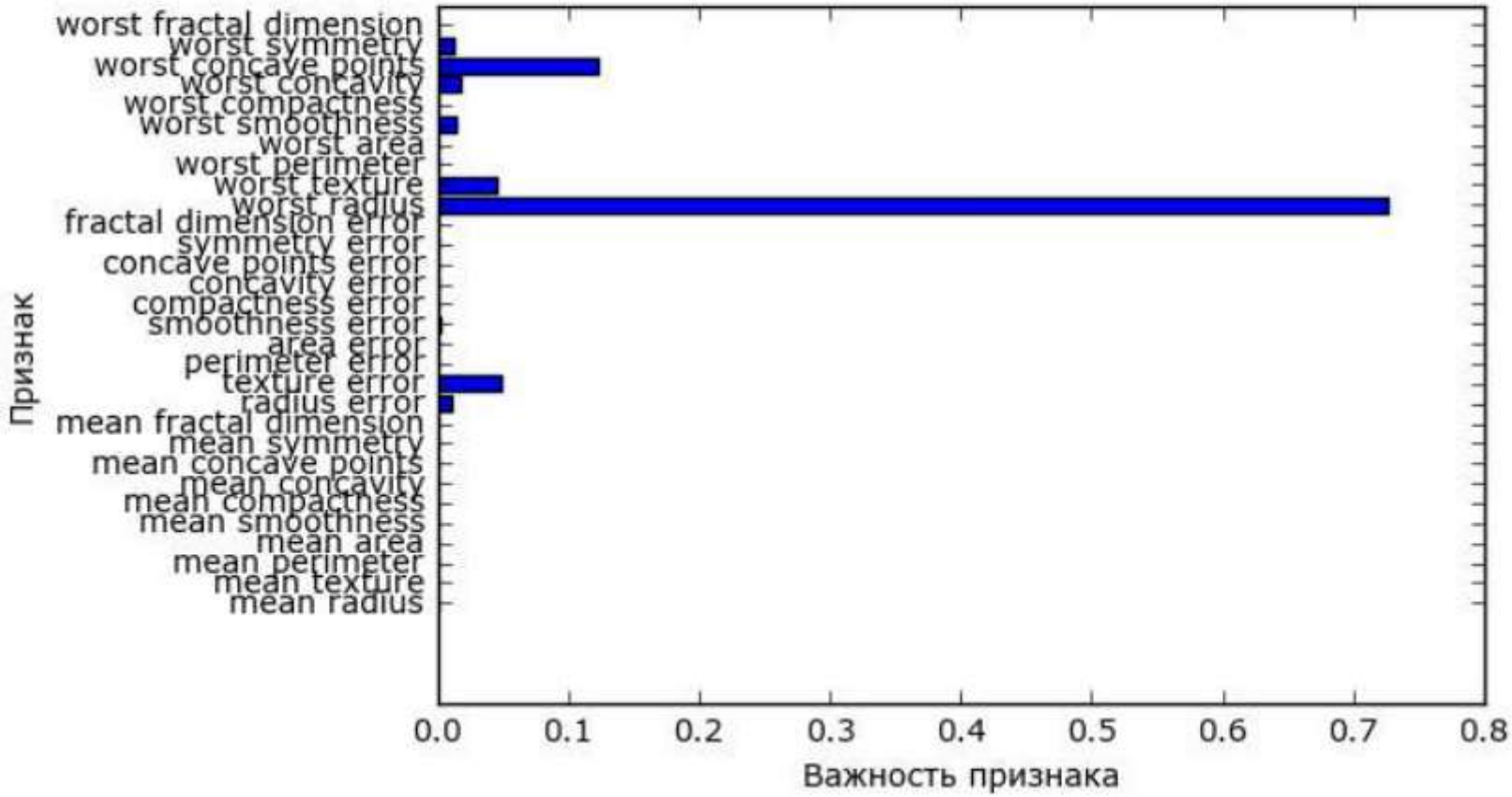
```
In[62]:
print("Важности признаков:\n{}".format(tree.feature_importances_))

Out[62]:
Важности признаков
[ 0.          0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.01019737  0.04839825  0.          0.
  0.0024156   0.          0.          0.          0.          0.
  0.72682851  0.0458159   0.          0.          0.0141577   0.          0.018188
  0.1221132   0.01188548  0.        ]
```

```python
for name, score in zip(cancer["feature_names"], tree.feature_importances_):
    print(name, score)
```

```
mean radius 0.0
mean texture 0.0
mean perimeter 0.0
mean area 0.0
mean smoothness 0.0
mean compactness 0.0
mean concavity 0.0
mean concave points 0.0
mean symmetry 0.0
mean fractal dimension 0.0
radius error 0.0101973682021
texture error 0.0483982536186
perimeter error 0.0
area error 0.0
smoothness error 0.00241559508532
compactness error 0.0
concavity error 0.0
concave points error 0.0
symmetry error 0.0
fractal dimension error 0.0
worst radius 0.72682850946
worst texture 0.0458158970889
worst perimeter 0.0
worst area 0.0
worst smoothness 0.0141577021047
worst compactness 0.0
worst concavity 0.0181879968645
worst concave points 0.122113199265
worst symmetry 0.0118854783101
worst fractal dimension 0.0
```

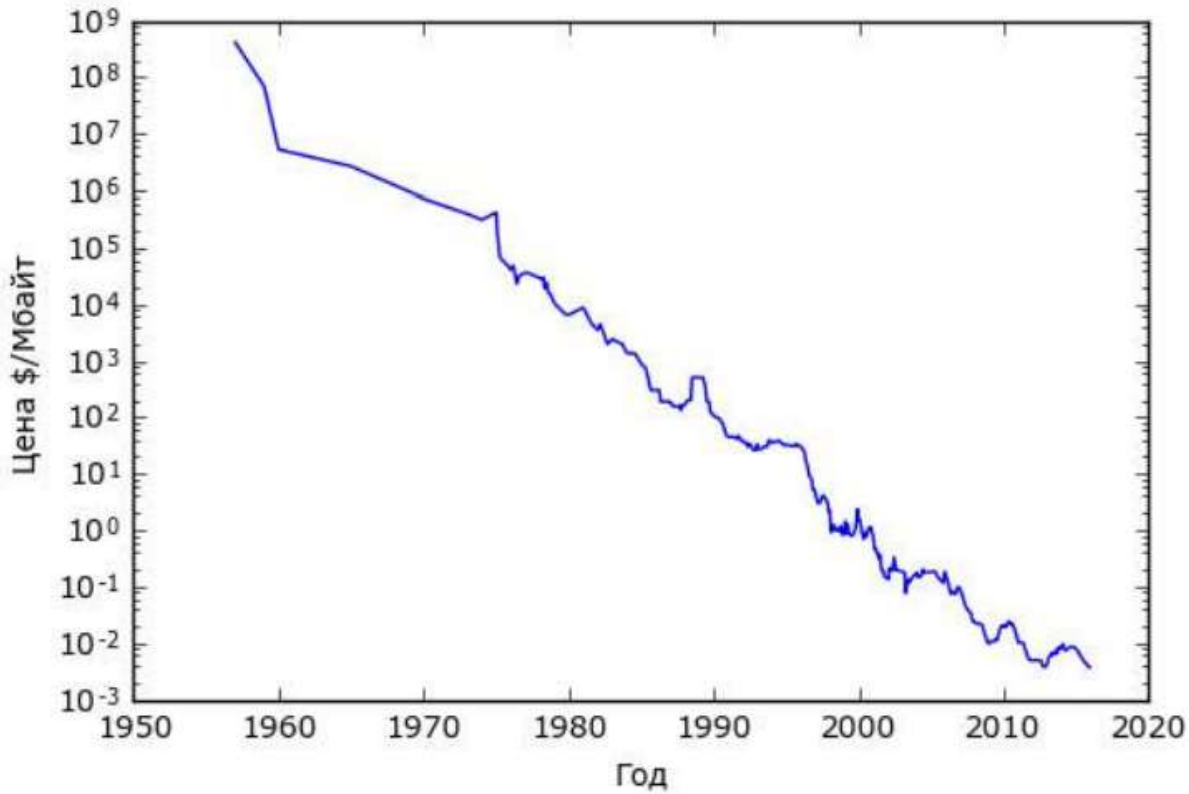
In[63]:

```
def plot_feature_importances_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)
    plt.xlabel("Важность признака")
    plt.ylabel("Признак")

plot_feature_importances_cancer(tree)
```

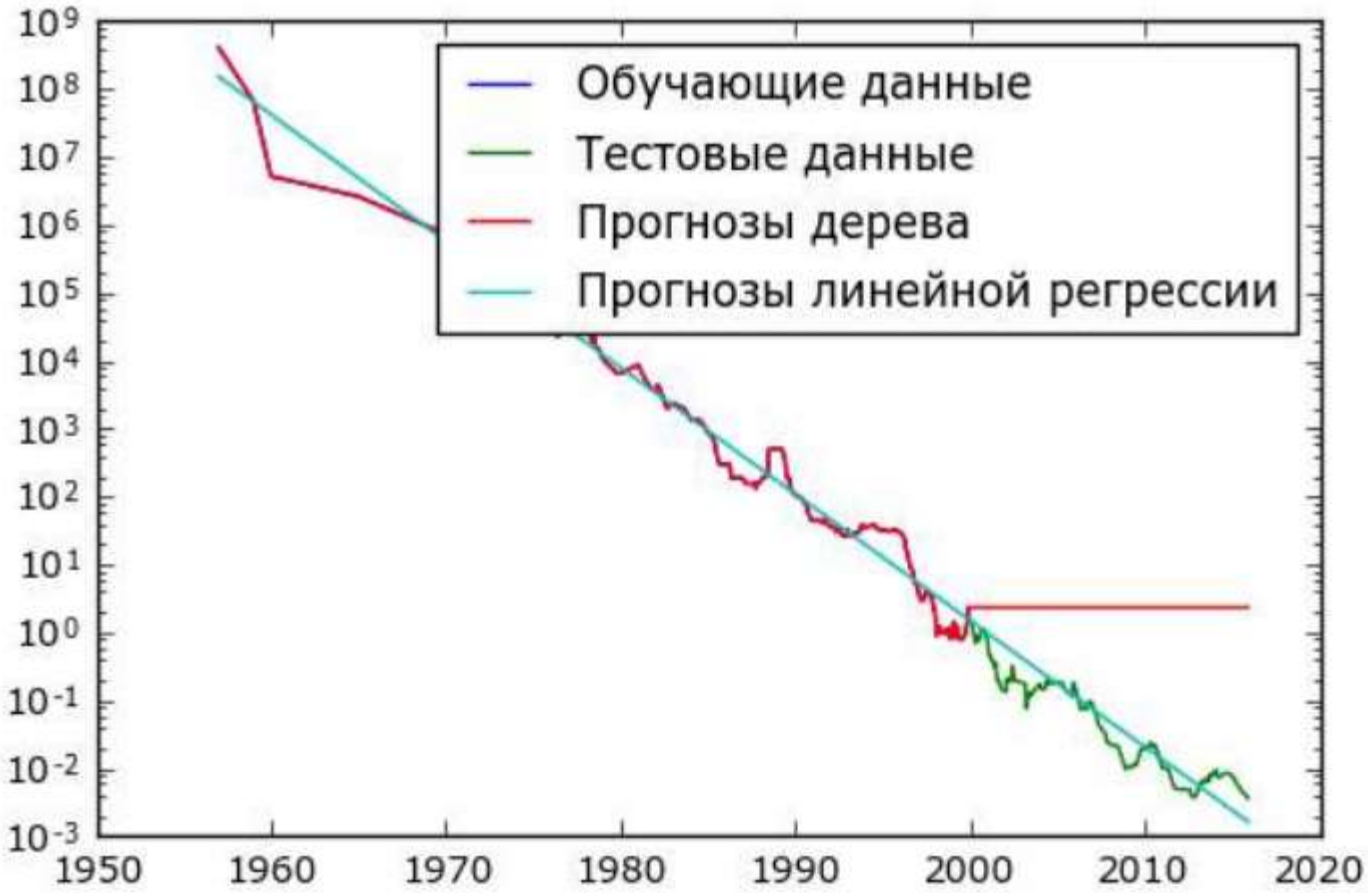


DecisionTreeRegressor

```
In[65]:
import pandas as pd
ram_prices = pd.read_csv("C:/Data/ram_price.csv")

plt.semilogy(ram_prices.date, ram_prices.price)
plt.xlabel("Год")
plt.ylabel("Цена $/Мбайт")
```



Comparing LR and DTR



Conclusion

- DT is easy in visualization
- Simple in understanding
- Not needed normalization and standardization of features