Домашнее задание к занятию 2.1: Деревья решений. Классификация

Обзор прошедшего занятия

Что мы делали в классе:

Задание 1

- строили деревья
- критерии информативности которых написали даже сами
- визуализировали границы принятия решений в 2d
- и рисовали сами деревья

Задание 2

- приняли участие в соревновании на Kaggle, переварив кучу текстовых фичей в численные, проведя кросс-валидацию и сделав сабмит

Задание 3

- построили руками несколько метрик качества бинарной классификации

Задание 4

- использовали их для оценки классификации разделения статей Ведомостей по топикам

дополнительно было много приятных ништяков. Например, облако слов, мультипоточность в целях парсинга, удобный инструмент для нахождения правильной css разметки, сохранение моделей в статичные файлы, разделение строк на слова и лемматизация этих слов

Домашнее задание

Lvl 1:

- взять подготовленные раннее данные из задачи **Titanic**, обучиться на них с помощью дерева решений и кросс-валидации и сделать сабмит
- кросс-валидацию желательно сделать сразу по нескольким фичам (параметр *grid* в *GridSearchCV*)
- определить самые важные фичи
- вывести дерево решений (можете попробовать установить pydot и webgraphviz для отрисовки деревьев внутри ноутбука)

Результат: скрины нового сабмита на Kaggle и построенного дерева

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.tree import DecisionTreeClassifier
```

In [2]:

```
train_ds = pd.read_csv('../titanic/train.csv')
test_ds = pd.read_csv('../titanic/test.csv')
```

In [3]:

```
train_ds.shape, test_ds.shape
```

Out[3]:

```
((891, 12), (418, 11))
```

Работа с данными

In [4]:

```
train_ds.columns
```

Out[4]:

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'S
ibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [5]:

```
from sklearn.preprocessing import LabelEncoder
```

In [6]:

```
def add_features(ds, is_train=False, cabin_encoder=None, embarked_encoder=None,
age_mode=None, fare_mode=None):
    ds['Sex_male'] = ds['Sex']=='male'
    ds['Cabin_letter'] = [x[0] if x is not np.nan else '-' for x in ds['Cabin']]
    if is_train:
        cabin_encoder = LabelEncoder()
        cabin_encoder.fit(ds['Cabin_letter'])
    else:
        pass
    ds['Cabin_letter_enc'] = cabin_encoder.transform(ds['Cabin_letter'])
    ds['Embarked'] = ds['Embarked'].fillna('-')
    if is_train:
        embarked_encoder = LabelEncoder()
        embarked_encoder.fit(ds['Embarked'])
    else:
        pass
    ds['Embarked_enc'] = embarked_encoder.transform(ds['Embarked'])
    age_mode = ds['Age'].mode()[0] if age_mode is None else age_mode
    ds['Age'] = ds['Age'].fillna(age_mode)
    fare_mode = ds['Fare'].mode()[0] if fare_mode is None else fare_mode
    ds['Fare'] = ds['Fare'].fillna(fare_mode)
    return ds, cabin_encoder, embarked_encoder, age_mode, fare_mode
```

In [7]:

```
train_ds.loc[train_ds['Embarked']=='0', 'Embarked'] = None
```

In [8]:

```
ds, cabin_encoder, embarked_encoder, age_mode, fare_mode = add_features(train_ds
, is_train=True)
add_features(test_ds, cabin_encoder=cabin_encoder, embarked_encoder=embarked_enc
oder, age_mode=age_mode, fare_mode=fare_mode);
```

Базовая модель

In [9]:

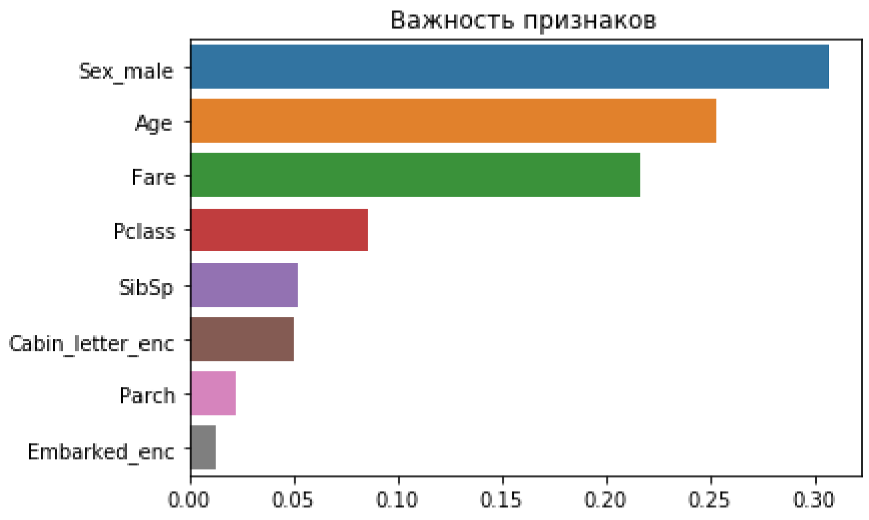
```
tree = DecisionTreeClassifier()
feats = ['Pclass', 'Sex_male','Age','SibSp','Parch','Fare','Cabin_letter_enc','E
mbarked_enc']
tree.fit(train_ds[feats], train_ds['Survived'])
```

Out[9]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_dept
h=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=None,
            splitter='best')
```

In [10]:

```
feat_imps = pd.Series(tree.feature_importances_, index=feats).sort_values(ascend
ing=False)
sns.barplot(x=feat_imps.values, y=feat_imps.index)
plt.title('Важность признаков')
plt.show()
```



Grid search

In [11]:

```
from sklearn.model_selection import GridSearchCV
```

In [12]:

```
grid = {
    'criterion':['gini','entropy'],
    'max_depth':[2,5,10,15,20,25,30,50,100,500,100],
    'min_samples_split': [2,5,10,50],
    'random_state':[42],
}
```

In [13]:

```
gs = GridSearchCV(DecisionTreeClassifier(), grid, cv=10)
gs.fit(train_ds[feats], train_ds['Survived'])
```

```
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/model_selec
tion/_search.py:841: DeprecationWarning: The default of the `iid` pa
rameter will change from True to False in version 0.22 and will be r
emoved in 0.24. This will change numeric results when test-set sizes
are unequal.
  DeprecationWarning)
```

Out[13]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=DecisionTreeClassifier(class_weight=None, criterion
='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=None,
            splitter='best'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'criterion': ['gini', 'entropy'], 'max_depth':
[2, 5, 10, 15, 20, 25, 30, 50, 100, 500, 100], 'min_samples_split':
[2, 5, 10, 50], 'random_state': [42]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='war
n',
       scoring=None, verbose=0)
```

In [14]:

```
gs_res = pd.DataFrame(gs.cv_results_)
print(gs_res.shape)
gs_res.head()
```

```
(88, 34)
```

```
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit0_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit1_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit2_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit3_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit4_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit5_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit6_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit7_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit8_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit9_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
```

```
cation.py:125: FutureWarning: You are accessing a training score ('m
ean_train_score'), which will not be available by default any more i
n 0.21. If you need training scores, please set return_train_score=T
rue
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
td_train_score'), which will not be available by default any more in
0.21. If you need training scores, please set return_train_score=Tru
e
  warnings.warn(*warn_args, **warn_kwargs)
```

Out[14]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_ma
0	0.003212	0.001629	0.000954	0.000329	gini	
1	0.002029	0.000011	0.000692	0.000006	gini	
2	0.002015	0.000012	0.000697	0.000007	gini	
3	0.002008	0.000009	0.000692	0.000007	gini	
4	0.002360	0.000019	0.000702	0.000005	gini	

5 rows × 34 columns

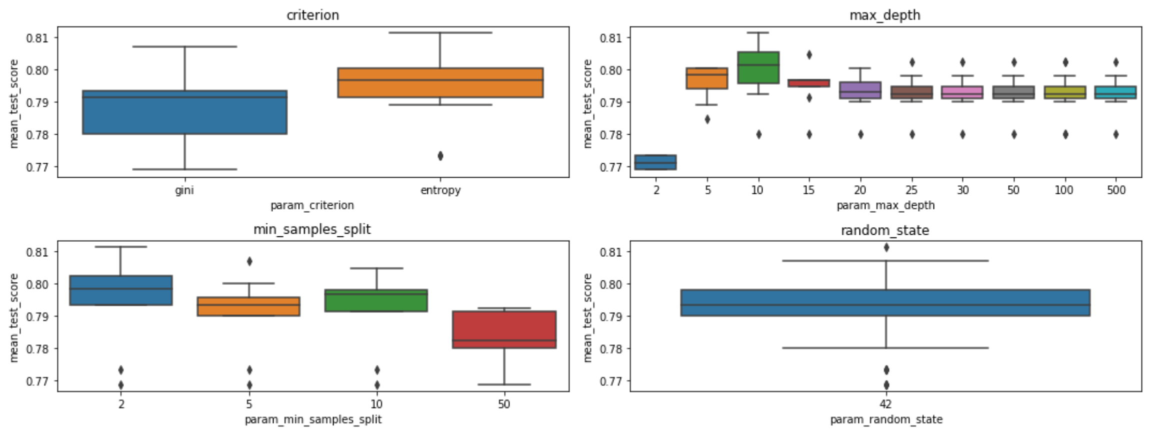
In [15]:

```
for f in [x for x in gs_res.columns if x.startswith('param_')]:
    print(f, gs_res[f].value_counts())
    print()
```

```
param_criterion gini       44
entropy    44
Name: param_criterion, dtype: int64

param_max_depth 100    16
15     8
30     8
10     8
25     8
5      8
20     8
500    8
2      8
50     8
Name: param_max_depth, dtype: int64

param_min_samples_split 5     22
10    22
2     22
50    22
Name: param_min_samples_split, dtype: int64

param_random_state 42    88
Name: param_random_state, dtype: int64
```

In [16]:

```
plt.figure(figsize=(15,8))
for idx, p in enumerate(grid.keys()):
    plt.subplot(len(grid)//2+1,2,idx+1)
    plt.title(p)
    #cds = gs_res.groupby('param_'+p)['mean_train_score'].mean()
    sns.boxplot(x='param_'+p, y='mean_test_score',data=gs_res)
plt.tight_layout()
plt.show()
```

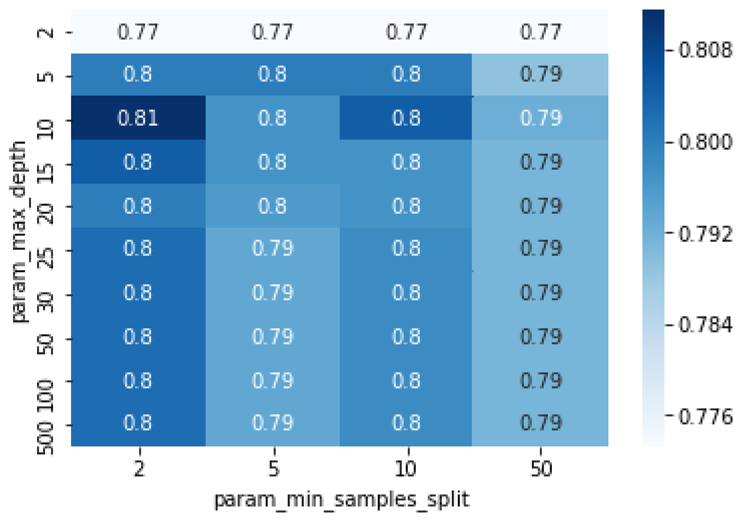


In [17]:

```
cds = gs_res[gs_res['param_criterion']=='entropy'].pivot_table(index='param_max_
depth',columns='param_min_samples_split', values='mean_test_score')
sns.heatmap(cds, cmap='Blues', annot=True)
```

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fde55a48a90>
```



In [18]:

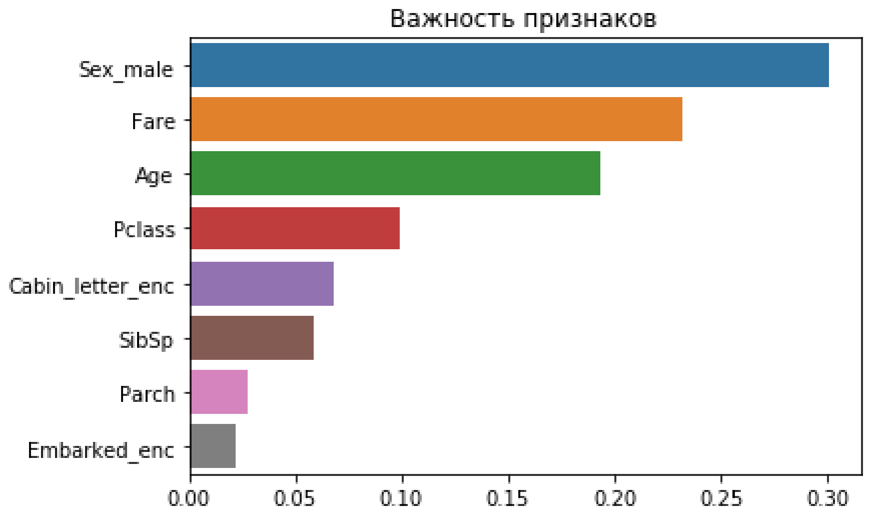
```
tree_gs = DecisionTreeClassifier(max_depth=10, min_samples_split=2,criterion='en
tropy',random_state=42)
tree_gs.fit(train_ds[feats], train_ds['Survived'])
```

Out[18]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_d
epth=10,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=42,
            splitter='best')
```

In [19]:

```
feat_imps = pd.Series(tree_gs.feature_importances_, index=feats).sort_values(asc
ending=False)
sns.barplot(x=feat_imps.values, y=feat_imps.index)
plt.title('Важность признаков')
plt.show()
```



Только важные признаки

In [20]:

```
imp_feats = ['Sex_male', 'Age','Fare']
```

In [21]:

```
grid = {
    'criterion':['gini','entropy'],
    'max_depth':[2,5,10,15,20,25,30,50,100,500,100],
    'min_samples_split': [2,5,10,50],
    'random_state':[42],
}
```

In [22]:

```
gs = GridSearchCV(DecisionTreeClassifier(), grid, cv=10)
gs.fit(train_ds[imp_feats], train_ds['Survived'])
```

Out[22]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=DecisionTreeClassifier(class_weight=None, criterion
='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=None,
            splitter='best'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'criterion': ['gini', 'entropy'], 'max_depth':
[2, 5, 10, 15, 20, 25, 30, 50, 100, 500, 100], 'min_samples_split':
[2, 5, 10, 50], 'random_state': [42]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='war
n',
       scoring=None, verbose=0)
```

In [23]:

```
gs_res = pd.DataFrame(gs.cv_results_)
print(gs_res.shape)
gs_res.head()
```

```
(88, 34)
```

```
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit0_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit1_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit2_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit3_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit4_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit5_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit6_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit7_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit8_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
plit9_train_score'), which will not be available by default any more
in 0.21. If you need training scores, please set return_train_score=
True
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
```

```
cation.py:125: FutureWarning: You are accessing a training score ('m
ean_train_score'), which will not be available by default any more i
n 0.21. If you need training scores, please set return_train_score=T
rue
  warnings.warn(*warn_args, **warn_kwargs)
/home/egor/anaconda3/lib/python3.7/site-packages/sklearn/utils/depre
cation.py:125: FutureWarning: You are accessing a training score ('s
td_train_score'), which will not be available by default any more in
0.21. If you need training scores, please set return_train_score=Tru
e
  warnings.warn(*warn_args, **warn_kwargs)
```

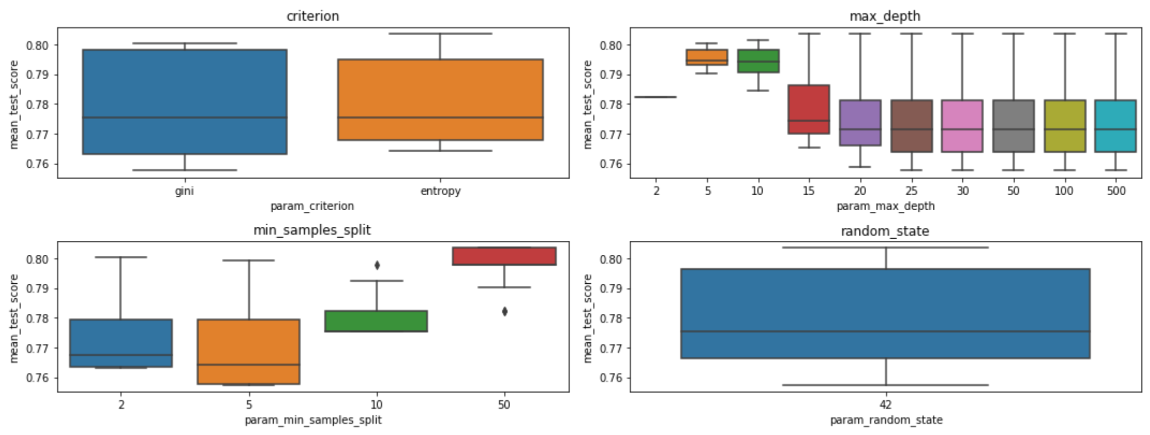
Out[23]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_ma
0	0.002938	0.001144	0.001070	0.000434	gini	
1	0.001707	0.000060	0.000673	0.000011	gini	
2	0.001674	0.000006	0.000662	0.000004	gini	
3	0.001678	0.000009	0.000667	0.000017	gini	
4	0.001919	0.000017	0.000669	0.000005	gini	

5 rows × 34 columns

In [24]:

```
plt.figure(figsize=(15,8))
for idx, p in enumerate(grid.keys()):
    plt.subplot(len(grid)//2+1,2,idx+1)
    plt.title(p)
    #cds = gs_res.groupby('param_'+p)['mean_train_score'].mean()
    sns.boxplot(x='param_'+p, y='mean_test_score',data=gs_res)
plt.tight_layout()
plt.show()
```

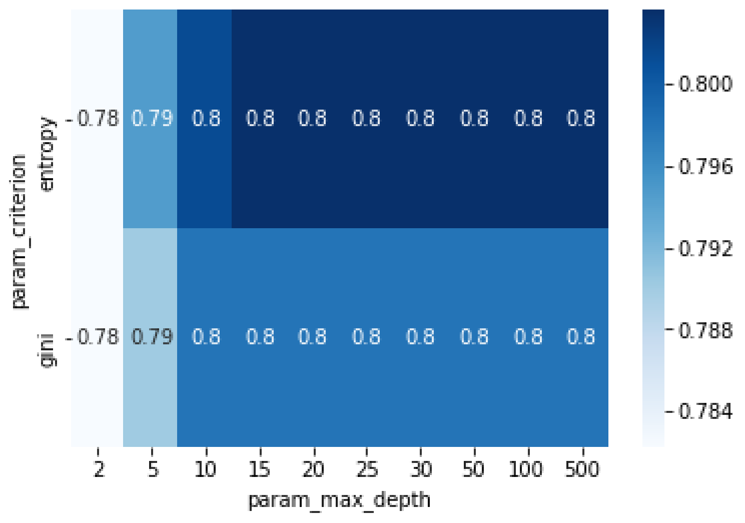


In [25]:

```
cds = gs_res[gs_res['param_min_samples_split']==50].pivot_table(index='param_cri
terion',columns='param_max_depth', values='mean_test_score')
sns.heatmap(cds, cmap='Blues', annot=True)
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fde55c91320>
```



In [26]:

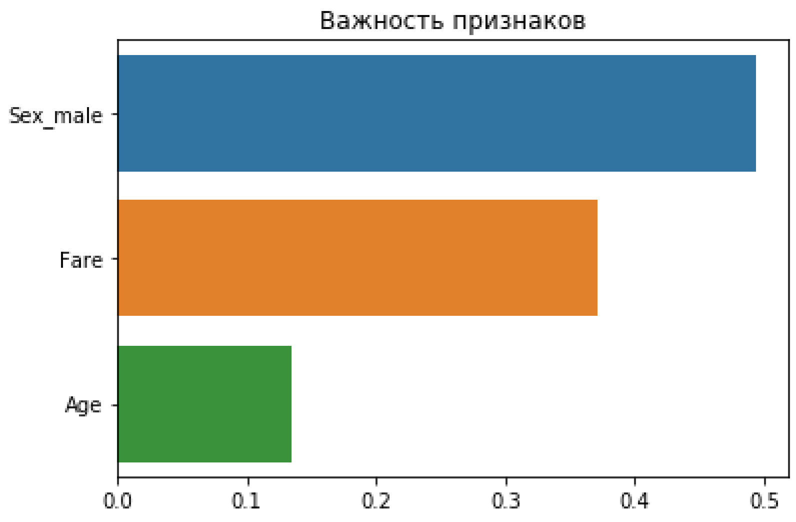
```
tree_gs_short = DecisionTreeClassifier(max_depth=15, min_samples_split=50,criter
ion='entropy',random_state=42)
tree_gs_short.fit(train_ds[imp_feats], train_ds['Survived'])
```

Out[26]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_d
epth=15,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=50,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=42,
            splitter='best')
```

In [27]:

```
feat_imps = pd.Series(tree_gs_short.feature_importances_, index=imp_feats).sort_
values(ascending=False)
sns.barplot(x=feat_imps.values, y=feat_imps.index)
plt.title('Важность признаков')
plt.show()
```



Визуализация дерева

In [28]:

```
import pydotplus
from sklearn.tree import export_graphviz
from IPython.display import Image
import re
```

Полное дерево

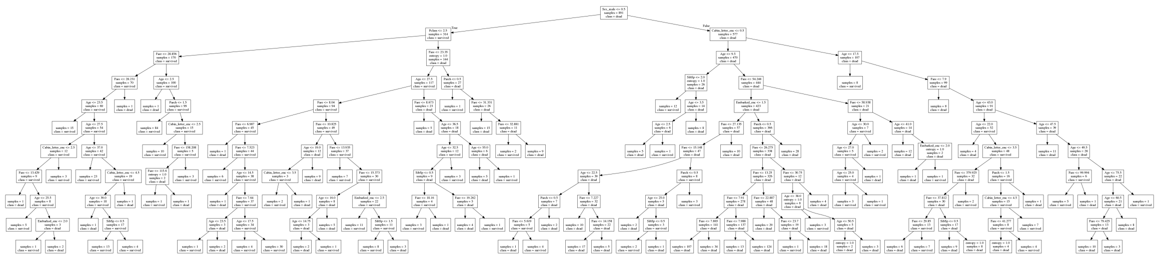
In [29]:

```
dot_data = export_graphviz(tree_gs, feature_names=feats, out_file=None, class_na
mes=['dead','survived'])

dot_data = re.sub(r'entropy = 0\.\d+', '', dot_data)
dot_data = re.sub(r'value = \[(\d|,| )+\]', '', dot_data)
dot_data = re.sub(r'\\n"', '"', dot_data)
dot_data = re.sub(r'(\\n)+', r'\\n', dot_data)

graph = pydotplus.graph_from_dot_data(dot_data)
```

In [30]:

```
Image(graph.create_png())
```

Out[30]:



Только важные признаки

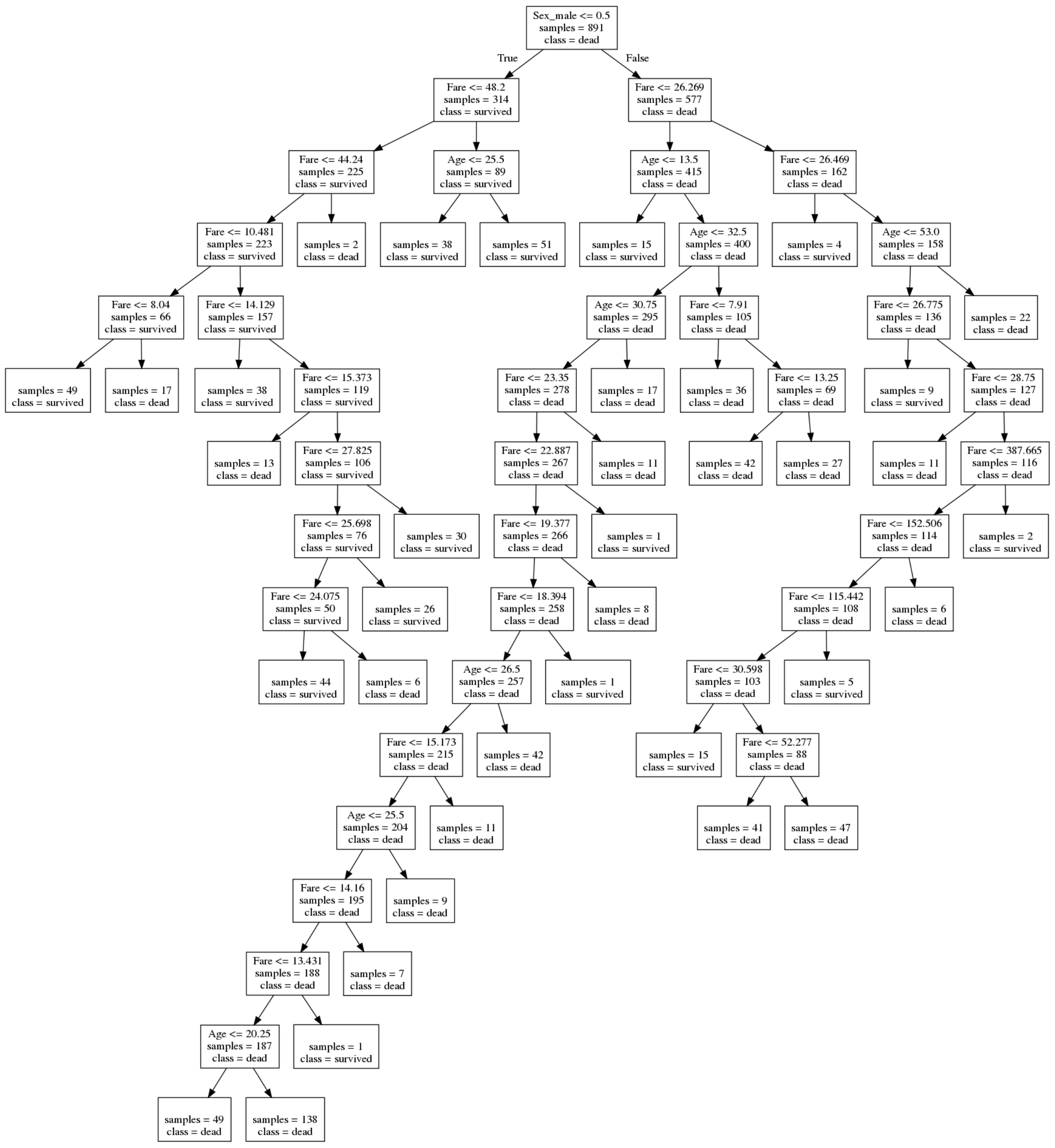
In [31]:

```
dot_data = export_graphviz(tree_gs_short, feature_names=imp_feats, out_file=None
, class_names=['dead','survived'])

dot_data = re.sub(r'entropy = 0\.\d+', '', dot_data)
dot_data = re.sub(r'value = \[(\d|,| )+\]', '', dot_data)
dot_data = re.sub(r'\\n"', '"', dot_data)
dot_data = re.sub(r'(\\n)+', r'\\n', dot_data)

graph = pydotplus.graph_from_dot_data(dot_data)
```

In [32]:

```
Image(graph.create_png())
```

Out[32]:


Sex_male <= 0.5 samples = 891 class = dead
True
False
Fare <= 48.2 samples = 314 class = survived
Fare <= 26.269 samples = 577 class = dead
Fare <= 44.24 samples = 225 class = survived
Age <= 25.5 samples = 89 class = survived
Age <= 13.5 samples = 415 class = dead
Fare <= 26.469 samples = 162 class = dead
Fare <= 10.481 samples = 223 class = survived
samples = 2 class = dead
samples = 38 class = survived
samples = 51 class = survived
samples = 15 class = survived
Age <= 32.5 samples = 400 class = dead
samples = 4 class = survived
Age <= 53.0 samples = 158 class = dead
Fare <= 8.04 samples = 66 class = survived
Fare <= 14.129 samples = 157 class = survived
Age <= 30.75 samples = 295 class = dead
Fare <= 7.91 samples = 105 class = dead
Fare <= 26.775 samples = 136 class = dead
samples = 22 class = dead
samples = 49 class = survived
samples = 17 class = dead
samples = 38 class = survived
Fare <= 15.373 samples = 119 class = survived
Fare <= 23.35 samples = 278 class = dead
samples = 17 class = dead
samples = 36 class = dead
Fare <= 13.25 samples = 69 class = dead
samples = 9 class = survived
Fare <= 28.75 samples = 127 class = dead
samples = 13 class = dead
Fare <= 27.825 samples = 106 class = survived
Fare <= 22.887 samples = 267 class = dead
samples = 11 class = dead
samples = 42 class = dead
samples = 27 class = dead
samples = 11 class = dead
Fare <= 387.665 samples = 116 class = dead
Fare <= 25.698 samples = 76 class = survived
samples = 30 class = survived
Fare <= 19.377 samples = 266 class = dead
samples = 1 class = survived
Fare <= 152.506 samples = 114 class = dead
samples = 2 class = survived
Fare <= 24.075 samples = 50 class = survived
samples = 26 class = survived
Fare <= 18.394 samples = 258 class = dead
samples = 8 class = dead
Fare <= 115.442 samples = 108 class = dead
samples = 6 class = dead
samples = 44 class = survived
samples = 6 class = dead
Age <= 26.5 samples = 257 class = dead
samples = 1 class = survived
Fare <= 30.598 samples = 103 class = dead
samples = 5 class = survived
Fare <= 15.173 samples = 215 class = dead
samples = 42 class = dead
samples = 15 class = survived
Fare <= 52.277 samples = 88 class = dead
Age <= 25.5 samples = 204 class = dead
samples = 11 class = dead
samples = 41 class = dead
samples = 47 class = dead
Fare <= 14.16 samples = 195 class = dead
samples = 9 class = dead
Fare <= 13.431 samples = 188 class = dead
samples = 7 class = dead
Age <= 20.25 samples = 187 class = dead
samples = 1 class = survived
samples = 49 class = dead
samples = 138 class = dead

In [42]:

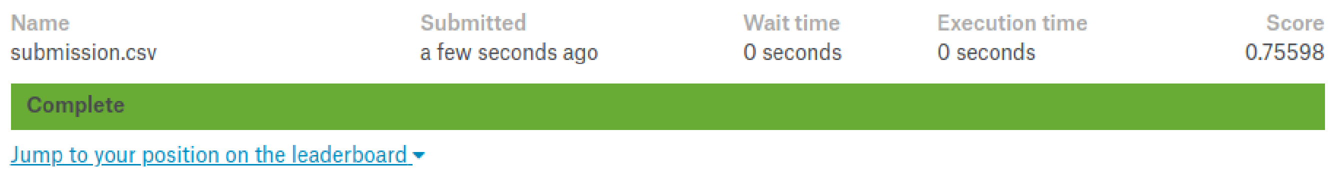
```
tree_gs_short.fit(train_ds[imp_feats], train_ds['Survived'])
test_ds['Preds'] = tree_gs_short.predict(test_ds[imp_feats])
```

In [53]:

```
to_kaggle = test_ds[['PassengerId','Preds']].rename(columns={'Preds':'Survived'
})
to_kaggle.to_csv('submission.csv', index=False)
to_kaggle.head()
```

Out[53]:

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1



Name	Submitted	Wait time	Execution time	Score
submission.csv	a few seconds ago	0 seconds	0 seconds	0.75598

Complete

Jump to your position on the leaderboard

In [56]:

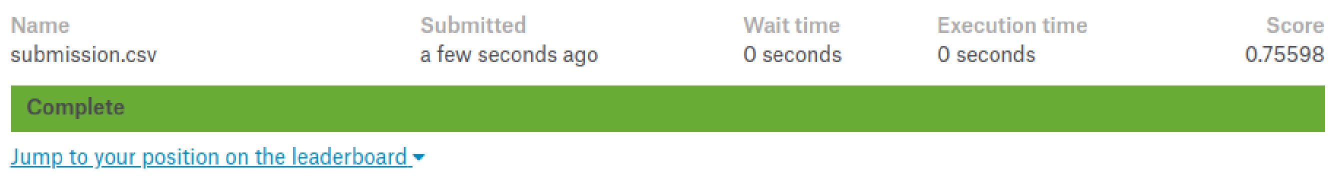
```
tree_gs.fit(train_ds[feats], train_ds['Survived'])
test_ds['Preds'] = tree_gs.predict(test_ds[feats])
```

In [57]:

```
to_kaggle = test_ds[['PassengerId','Preds']].rename(columns={'Preds':'Survived'
})
to_kaggle.to_csv('submission_gs.csv', index=False)
to_kaggle.head()
```

Out[57]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	1



Name	Submitted	Wait time	Execution time	Score
submission.csv	a few seconds ago	0 seconds	0 seconds	0.75598

Complete

Jump to your position on the leaderboard

In [58]:

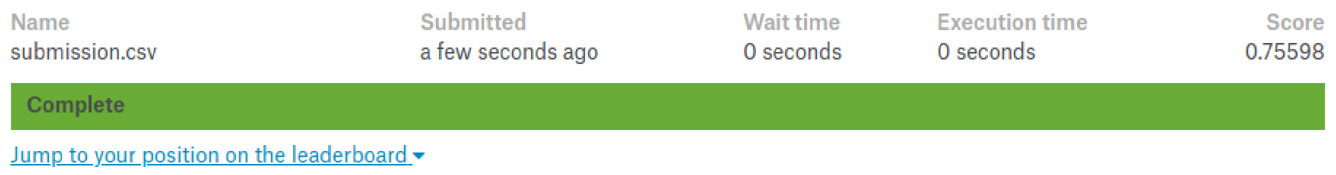
```
tree.fit(train_ds[feats], train_ds['Survived'])
test_ds['Preds'] = tree.predict(test_ds[feats])
```

In [59]:

```
to_kaggle = test_ds[['PassengerId','Preds']].rename(columns={'Preds':'Survived'
})
to_kaggle.to_csv('submission_base.csv', index=False)
to_kaggle.head()
```

Out[59]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	1
3	895	1
4	896	1



Name	Submitted	Wait time	Execution time	Score
submission.csv	a few seconds ago	0 seconds	0 seconds	0.75598

Complete

Jump to your position on the leaderboard

Lvl 2: (опционально)

- С помощью функций sklearn.metrics.auc, precision, recall составить функцию для расчёта ROC-AUC, ROC-PRC
- Придумать себе интересную задачу на основе данных из интернета =) Спарсить ещё какой-нибудь сайт (не Ведомости) и решить задачу классификации. Делать свои проекты - круто. Если будут - кидайте мне =) @NikitaKuznetsov (http://t.me/NikitaKuznesov)

In [33]:

```
from sklearn.metrics import auc, precision_score, recall_score
```

In [34]:

```
def get_ROC_PRC(probs, real):
    threshs = np.linspace(0.01,0.99,100)
    precs = [precision_score(real, probs>t) for t in threshs]
    recs = [recall_score(real, probs>t) for t in threshs]
    precs = [0]+precs+[1]
    recs = [1]+recs+[0]
    auc_prc = auc(recs, precs)
    return precs, recs, auc_prc
```

In [35]:

```
n_train = int(len(train_ds)*0.8)
```

In [36]:

```
tree.fit(train_ds[feats][:n_train], train_ds['Survived'][:n_train])
tree_gs.fit(train_ds[feats][:n_train], train_ds['Survived'][:n_train])
tree_gs_short.fit(train_ds[imp_feats][:n_train], train_ds['Survived'][:n_train
]);
```

In [37]:

```
train_ds['Pred'] = tree.predict_proba(train_ds[feats])[:,1]
precs, recs, auc_prc = get_ROC_PRC(train_ds['Pred'][n_train:], train_ds['Survive
d'][n_train:])
auc_prc
```

Out[37]:

```
0.7611425650423213
```

In [38]:

```
train_ds['Pred'] = tree_gs.predict_proba(train_ds[feats])[:,1]
precs_gs, recs_gs, auc_prc_gs = get_ROC_PRC(train_ds['Pred'][n_train:], train_ds
['Survived'][n_train:])
auc_prc_gs
```

Out[38]:

```
0.7670211388770913
```

In [39]:

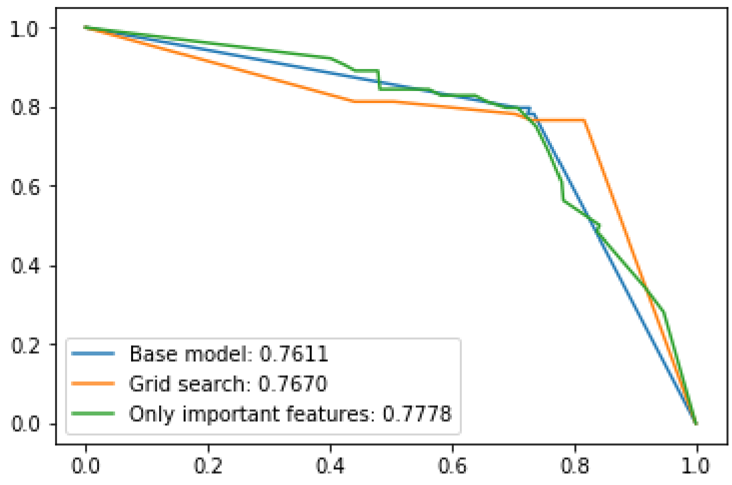
```
train_ds['Pred'] = tree_gs_short.predict_proba(train_ds[imp_feats])[:,1]
precs_gs_short, recs_gs_short, auc_prc_gs_short = get_ROC_PRC(train_ds['Pred'][n
_train:], train_ds['Survived'][n_train:])
auc_prc_gs_short
```

Out[39]:

```
0.7777891685515299
```

In [40]:

```
plt.plot(precs, recs, label=f'Base model: {auc_prc:.4f}')
plt.plot(precs_gs, recs_gs, label=f'Grid search: {auc_prc_gs:.4f}')
plt.plot(precs_gs_short, recs_gs_short, label=f'Only important features: {auc_pr
c_gs_short:.4f}')
plt.legend()
plt.show()
```


Base model: 0.7611
Grid search: 0.7670
Only important features: 0.7778