

Лабораторная работа 5
по дисциплине
«Методы машинного обучения»
на тему
«Линейные модели, SVM и деревья решений.»

Выполнил:
студент группы ИУ5-21М
Могильников И. А.

1. Описание задания

Цель лабораторной работы: изучение линейных моделей, SVM и деревьев решений.

1.1. Задание

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода `train_test_split` разделите выборку на обучающую и тестовую.
4. Обучите 1) одну из линейных моделей, 2) SVM и 3) дерево решений. Оцените качество моделей с помощью трех подходящих для задачи метрик. Сравните качество полученных моделей.
5. Произведите для каждой модели подбор одного гиперпараметра с использованием `GridSearchCV` и кросс-валидации.
6. Повторите пункт 4 для найденных оптимальных значений гиперпараметров. Сравните качество полученных моделей с качеством моделей, полученных в пункте 4.

2. Ход выполнения лабораторной работы

2.1. Выбор датасета

В качестве исходных данных выбираем датасет Heart Disease UCI (<https://www.kaggle.com/ronitf/heart-disease-uci>). 303 записи, 14 признаков, целевой признак относится к наличию болезни сердца у пациента: 0 - нет болезни сердца, 1 - есть.

```
In [0]: from google.colab import drive, files
        drive.mount('/content/drive')
```

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189898

Enter your authorization code:

.....

Mounted at /content/drive

```
In [0]: from google.colab import files
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
os.listdir()
data = pd.read_csv('drive/My Drive/mmo_datasets/heart.csv',
                  sep=";", encoding="iso-8859-1")
```

```
In [0]: total_count = data.shape[0]
        num_cols = []
        for col in data.columns:
            # Количество пустых значений
            temp_null_count = data[data[col].isnull()].shape[0]
            dt = str(data[col].dtype)
            if temp_null_count>0:
                num_cols.append(col)
                temp_perc = round((temp_null_count / total_count) * 100.0, 2)
                print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'
                      .format(col, dt, temp_null_count, temp_perc))
```

```
data_cleared = data
```

```
In [0]: uniquevalues = np.unique(data_cleared['target'].values)
        uniquevalues
```

```
Out[0]: array([0, 1])
```

2.2. train_test_split

```
In [0]: target = data_cleared['target']
        data_cleared = data_cleared.drop('target', axis=1)
```

```
In [0]: data_cleared.head(10)
```

```
Out[0]:
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope \
0	63	1	3	145	233	1	0	150	0	2.3	0
1	37	1	2	130	250	0	1	187	0	3.5	0
2	41	0	1	130	204	0	0	172	0	1.4	2
3	56	1	1	120	236	0	1	178	0	0.8	2
4	57	0	0	120	354	0	1	163	1	0.6	2
5	57	1	0	140	192	0	1	148	0	0.4	1
6	56	0	1	140	294	0	0	153	0	1.3	1
7	44	1	1	120	263	0	1	173	0	0.0	2
8	52	1	2	172	199	1	1	162	0	0.5	2
9	57	1	2	150	168	0	1	174	0	1.6	2

	ca	thal
0	0	1
1	0	2
2	0	2
3	0	2
4	0	2
5	0	1
6	0	2
7	0	3
8	0	3
9	0	2

```
In [0]: from sklearn.model_selection import train_test_split
        X_train, X_test, Y_train, Y_test = train_test_split(
```

```
data_cleared,  
target,  
test_size=0.2,  
random_state=1  
)
```

```
In [0]: X_train.shape, Y_train.shape
```

```
Out[0]: ((242, 13), (242,))
```

```
In [0]: X_test.shape, Y_test.shape
```

```
Out[0]: ((61, 13), (61,))
```

2.3. Обучение

```
In [0]: from sklearn.linear_model import SGDClassifier  
        from sklearn.svm import LinearSVC  
        from sklearn.tree import DecisionTreeClassifier  
        from sklearn.model_selection import GridSearchCV  
  
        from sklearn.metrics import accuracy_score  
        from sklearn.metrics import balanced_accuracy_score  
        from sklearn.metrics import precision_score, recall_score, f1_score
```

2.3.1. Стохастический градиентный спуск

```
In [0]: sgd = SGDClassifier().fit(X_train, Y_train)  
        predicted_sgd = sgd.predict(X_test)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:166: FutureWarning
```

```
In [0]: accuracy_score(Y_test, predicted_sgd)
```

```
Out[0]: 0.5081967213114754
```

```
In [0]: balanced_accuracy_score(Y_test, predicted_sgd)
```

```
Out[0]: 0.5161290322580645
```

```
In [0]: (precision_score(Y_test, predicted_sgd, average='weighted'),  
        recall_score(Y_test, predicted_sgd, average='weighted'))
```

```
Out[0]: (0.7540983606557377, 0.5081967213114754)
```

```
In [0]: f1_score(Y_test, predicted_sgd, average='weighted')
```

```
Out[0]: 0.35963114754098363
```

2.3.2. Линейный классификатор на основе SVM

```
In [0]: svm = LinearSVC(C=1.0).fit(X_train, Y_train)
        predicted_svm = svm.predict(X_test)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/svm/base.py:931: ConvergenceWarning: Liblinear
"the number of iterations.", ConvergenceWarning)
```

```
In [0]: accuracy_score(Y_test, predicted_svm)
```

```
Out[0]: 0.7704918032786885
```

```
In [0]: balanced_accuracy_score(Y_test, predicted_svm)
```

```
Out[0]: 0.7677419354838709
```

```
In [0]: (precision_score(Y_test, predicted_svm, average='weighted'),
        recall_score(Y_test, predicted_svm, average='weighted'))
```

```
Out[0]: (0.8020791683326669, 0.7704918032786885)
```

```
In [0]: f1_score(Y_test, predicted_svm, average='weighted')
```

```
Out[0]: 0.7634790528233151
```

2.3.3. Дерево решений

```
In [0]: dt = DecisionTreeClassifier(random_state=1).fit(X_train, Y_train)
        predicted_dt = dt.predict(X_test)
```

```
In [0]: accuracy_score(Y_test, predicted_dt)
```

```
Out[0]: 0.6885245901639344
```

```
In [0]: balanced_accuracy_score(Y_test, predicted_dt)
```

```
Out[0]: 0.6887096774193548
```

```
In [0]: (precision_score(Y_test, predicted_dt, average='weighted'),
        recall_score(Y_test, predicted_dt, average='weighted'))
```

```
Out[0]: (0.6888947646747752, 0.6885245901639344)
```

```
In [0]: f1_score(Y_test, predicted_dt, average='weighted')
```

```
Out[0]: 0.6885245901639343
```

Из двух представленных моделей с параметрами по умолчанию с задачей классификации на выбранном датасете лучше справляется линейный классификатор на основе SVM.

2.4. Подбор гиперпараметров

2.4.1. Стохастический градиентный спуск

```
In [0]: n_range = np.array(range(0,100,5))  
        n_range = n_range / 100  
        tuned_parameters = [{'l1_ratio': n_range}]  
        tuned_parameters
```

```
Out[0]: [{'l1_ratio': array([0. , 0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 ,  
0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95])}]
```

```
In [0]: import warnings  
        warnings.filterwarnings('ignore')
```

```
clf_gs_sgd = GridSearchCV(SGDClassifier(), tuned_parameters, cv=5,  
                           scoring='accuracy')  
clf_gs_sgd.fit(X_train, Y_train)
```

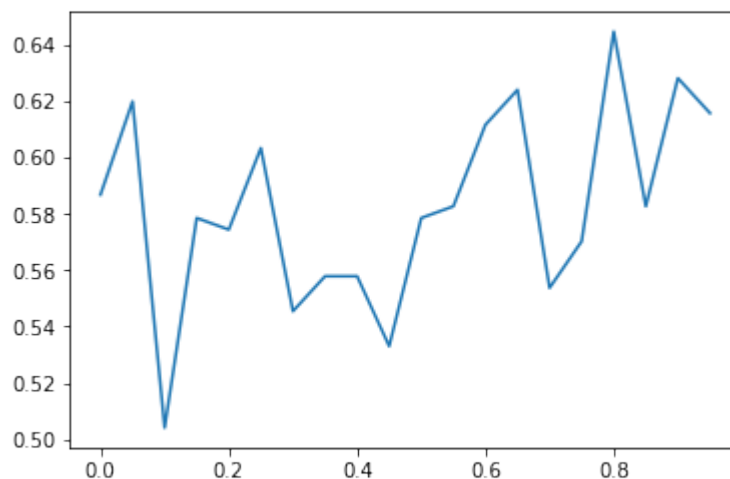
```
Out[0]: GridSearchCV(cv=5, error_score='raise-deprecating',  
    estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,  
    early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,  
    l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,  
    n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',  
    power_t=0.5, random_state=None, shuffle=True, tol=None,  
    validation_fraction=0.1, verbose=0, warm_start=False),  
    fit_params=None, iid='warn', n_jobs=None,  
    param_grid=[{'l1_ratio': array([0. , 0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 ,  
    0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95])}],  
    pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',  
    scoring='accuracy', verbose=0)
```

```
In [0]: clf_gs_sgd.best_params_
```

```
Out[0]: {'l1_ratio': 0.8}
```

```
In [0]: plt.plot(n_range, clf_gs_sgd.cv_results_['mean_test_score'])
```

```
Out[0]: [<matplotlib.lines.Line2D at 0x7f3fe94ec048>]
```



2.4.2. Линейный классификатор на основе SVM

```
In [0]: n_range = np.array(range(1,20,1))  
        tuned_parameters = [{'C': n_range}]  
        tuned_parameters
```

```
Out[0]: [{'C': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,  
                      18, 19])}]
```

```
In [0]: clf_gs_svm = GridSearchCV(LinearSVC(), tuned_parameters, cv=3,  
                                   scoring='accuracy')  
        clf_gs_svm.fit(X_train, Y_train)
```

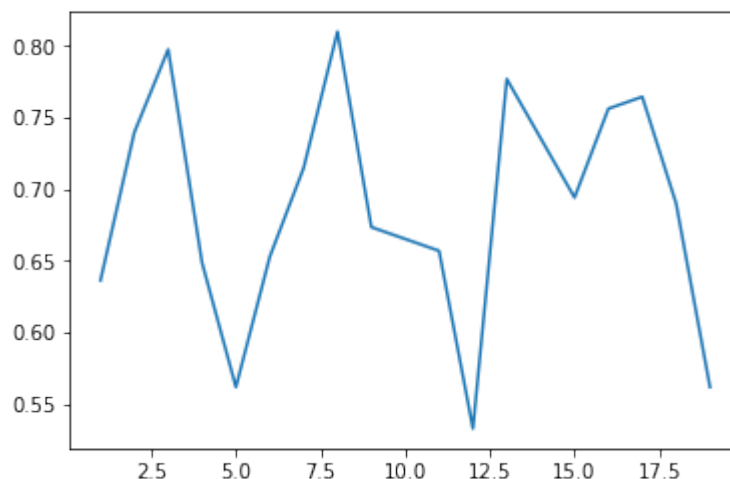
```
Out[0]: GridSearchCV(cv=3, error_score='raise-deprecating',  
                    estimator=LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,  
                    intercept_scaling=1, loss='squared_hinge', max_iter=1000,  
                    multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,  
                    verbose=0),  
                    fit_params=None, iid='warn', n_jobs=None,  
                    param_grid=[{'C': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,  
                    18, 19])}],  
                    pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',  
                    scoring='accuracy', verbose=0)
```

```
In [0]: clf_gs_svm.best_params_
```

```
Out[0]: {'C': 8}
```

```
In [0]: plt.plot(n_range, clf_gs_svm.cv_results_['mean_test_score'])
```

```
Out[0]: [<matplotlib.lines.Line2D at 0x7f3fe94c5dd8>]
```



2.4.3. Дерево решений

```
In [0]: n_range = np.array(range(1,7,1))  
        tuned_parameters = [{'max_depth': n_range}]  
        tuned_parameters
```

```
Out[0]: [{'max_depth': array([1, 2, 3, 4, 5, 6])}]
```

```
In [0]: clf_gs_dt = GridSearchCV(DecisionTreeClassifier(random_state=1), tuned_parameters,  
                                cv=5, scoring='accuracy')  
        clf_gs_dt.fit(X_train, Y_train)
```

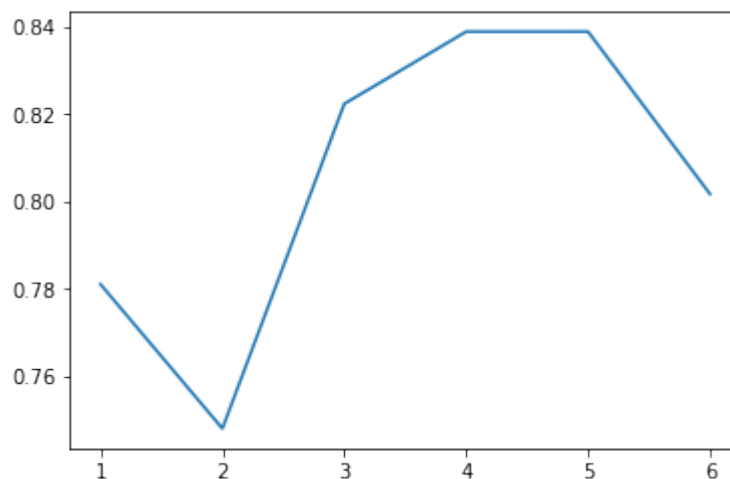
```
Out[0]: GridSearchCV(cv=5, error_score='raise-deprecating',  
                    estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=N  
                    max_features=None, max_leaf_nodes=None,  
                    min_impurity_decrease=0.0, min_impurity_split=None,  
                    min_samples_leaf=1, min_samples_split=2,  
                    min_weight_fraction_leaf=0.0, presort=False, random_state=1,  
                    splitter='best'),  
                    fit_params=None, iid='warn', n_jobs=None,  
                    param_grid=[{'max_depth': array([1, 2, 3, 4, 5, 6])}],  
                    pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',  
                    scoring='accuracy', verbose=0)
```

```
In [0]: clf_gs_dt.best_params_
```

```
Out[0]: {'max_depth': 4}
```

```
In [0]: plt.plot(n_range, clf_gs_dt.cv_results_['mean_test_score'])
```

```
Out[0]: [<matplotlib.lines.Line2D at 0x7f3fe9368dd8>]
```



2.5. Сравнение моделей после подбора гиперпараметров

2.5.1. Стохастический градиентный спуск

```
In [0]: sgd_optimized = SGDClassifier(l1_ratio=clf_gs_sgd.best_params_['l1_ratio']).fit(X_train, Y_train)
        predicted_sgd_opt = sgd_optimized.predict(X_test)
```

```
In [0]: accuracy_score(Y_test, predicted_sgd_opt)
```

```
Out[0]: 0.5409836065573771
```

```
In [0]: balanced_accuracy_score(Y_test, predicted_sgd_opt)
```

```
Out[0]: 0.5338709677419355
```

```
In [0]: (precision_score(Y_test, predicted_sgd_opt, average='weighted'),
        recall_score(Y_test, predicted_sgd_opt, average='weighted'))
```

```
Out[0]: (0.6363244176013804, 0.5409836065573771)
```

```
In [0]: f1_score(Y_test, predicted_sgd_opt, average='weighted')
```

```
Out[0]: 0.4332865784167616
```

2.5.2. Линейный классификатор на основе SVM

```
In [0]: svm_optimized = LinearSVC(C=clf_gs_svm.best_params_['C']).fit(X_train, Y_train)
        predicted_svm_opt = svm_optimized.predict(X_test)
```

```
In [0]: accuracy_score(Y_test, predicted_svm_opt)
```

```
Out[0]: 0.6885245901639344
```

```
In [0]: balanced_accuracy_score(Y_test, predicted_svm_opt)
```

```
Out[0]: 0.6833333333333333
```

```
In [0]: (precision_score(Y_test, predicted_svm_opt, average='weighted'),
        recall_score(Y_test, predicted_svm_opt, average='weighted'))
```

```
Out[0]: (0.8068852459016393, 0.6885245901639344)
```

```
In [0]: f1_score(Y_test, predicted_svm_opt, average='weighted')
```

```
Out[0]: 0.6528845252022647
```

2.5.3. Дерево решений

```
In [0]: dt_optimized = DecisionTreeClassifier(max_depth=clf_gs_dt.best_params_['max_depth'])  
        predicted_dt_opt = dt_optimized.predict(X_test)
```

```
In [0]: accuracy_score(Y_test, predicted_dt_opt)
```

```
Out[0]: 0.7704918032786885
```

```
In [0]: balanced_accuracy_score(Y_test, predicted_dt_opt)
```

```
Out[0]: 0.7704301075268818
```

```
In [0]: (precision_score(Y_test, predicted_dt_opt, average='weighted'),  
        recall_score(Y_test, predicted_dt_opt, average='weighted'))
```

```
Out[0]: (0.7704918032786885, 0.7704918032786885)
```

```
In [0]: f1_score(Y_test, predicted_dt_opt, average='weighted')
```

```
Out[0]: 0.7704918032786885
```

Подбор гиперпараметров позволил увеличить точность работы стохастического градиентного спуска и дерева решений. В случае с деревом решений, точность модели увеличилась существенно и после подбора гиперпараметров именно эта модель предоставляет наибольшую точность.