Министерство образования и науки Российской Федерации

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н.Э. БАУМАНА

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления» (ИУ5)

ДИСЦИПЛИНА: «Технологии машинного обучения»

Отчет по лабораторной работе №5

«Линейные модели, SVM и деревья решений»

Выполнил:

Студент группы ИУ5-61Б

Кочетков М.Д.

Преподаватель:

Гапанюк Ю.Е.

Москва, 2020 г.

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

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 следующие модели:
   - одну из линейных моделей;
   - SVM;
   - дерево решений.
5. Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

Выполненная работа:

В данной лабораторной работе модели будут строиться для решения задачи классификации. Загрузка и первичный анализ данных. Формирование DataFrame:

In [2]:
```
# используем выборку, связанную с вином
wine = load_wine()
```

In [3]:
```
# Сформируем DataFrame
wine_df = pd.DataFrame(data= np.c_[wine['data'], wine['target']],
                       columns= list(wine['feature_names']) + ['target'])
```

In [4]:
```
# проверим на пустые значения
wine_df.isnull().sum()
```

Out[4]:
```
alcohol                         0
malic_acid                      0
ash                             0
alcalinity_of_ash               0
magnesium                       0
total_phenols                   0
flavanoids                      0
nonflavanoid_phenols            0
proanthocyanins                 0
color_intensity                 0
hue                             0
od280/od315_of_diluted_wines    0
proline                         0
target                          0
dtype: int64
```

Разделение данных на обучающую и тестовую выборки. Построение модели «Логистическая регрессия»:

In [5]:
```
def convert_target_to_binary(array:np.ndarray, target:int) -> np.ndarray:
# Если целевой признак совпадает с указанным, то 1 иначе 0
    res = [1 if x==target else 0 for x in array]
    return res
```

In [6]:
```
bin_wine_y = convert_target_to_binary(wine.target, 2)
```

In [7]:
```
wine_X_train, wine_X_test, wine_y_train, wine_y_test = train_test_split(
    wine.data, bin_wine_y, test_size=0.3, random_state=1)
```

In [8]:
```
cl1 = LogisticRegression()
```

In [9]:
```
cl1.fit(wine_X_train, wine_y_train)
```

```
/usr/local/anaconda3/envs/ml/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:938: ConvergenceWarning: lbfgs faile
d to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[9]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [10]:
```
pred_wine_y_test = cl1.predict(wine_X_test)
pred_wine_y_test
```

Out[10]:
```
array([1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0])
```

In [11]:
```
pred_wine_y_test_proba = cl1.predict_proba(wine_X_test)
pred_wine_y_test_proba[:10]
```

Out[11]:
```
array([[0.06973962, 0.93026038],
       [0.99880988, 0.00119012],
       [0.99846945, 0.00153055],
       [0.99892742, 0.00107258],
       [0.98747465, 0.01252535],
       [0.04702843, 0.95297157],
       [0.95064996, 0.04935004],
       [0.99878499, 0.00121501],
       [0.00740072, 0.99259928],
       [0.96261773, 0.03738227]])
```

In [12]:
```
# Вероятность принадлежности к 0 классу
[round(x, 4) for x in pred_wine_y_test_proba[:10,0]]
```

Out[12]:
```
[0.0697, 0.9988, 0.9985, 0.9989, 0.9875, 0.047, 0.9506, 0.9988, 0.0074, 0.9626]
```

In [13]:
```
# Вероятность принадлежности к 1 классу
[round(x, 4) for x in pred_wine_y_test_proba[:10,1]]
```

Out[13]:
```
[0.9303, 0.0012, 0.0015, 0.0011, 0.0125, 0.953, 0.0494, 0.0012, 0.9926, 0.0374]
```

In [14]:
```
# Сумма вероятностей равна 1
pred_wine_y_test_proba[:10,0] + pred_wine_y_test_proba[:10,1]
```

Out[14]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [15]:
```
accuracy_score(wine_y_test, pred_wine_y_test)
```

Out[15]:
```
0.9814814814814815
```

In [16]:
```
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    """
    Вычисление метрики accuracy для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Accuracy для данного класса
    """
    # Для удобства фильтрации сформируем Pandas DataFrame
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
    # Метки классов
    classes = np.unique(y_true)
    # Результирующий словарь
    res = dict()
    # Перебор меток классов
    for c in classes:
        # отфильтруем данные, которые соответствуют
        # текущей метке класса в истинных значениях
        temp_data_flt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res
```

In [17]:
```
def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
        print('{} \t {}'.format(i, accs[i]))
```

In [18]:
```
print_accuracy_score_for_classes(wine_y_test, pred_wine_y_test)
```

```
Метка    Accuracy
0        1.0
1        0.9166666666666666
```

Построение модели «SVC»:

In [19]:
```
wine_X = wine.data[:, :2]
wine_y = wine.target
```

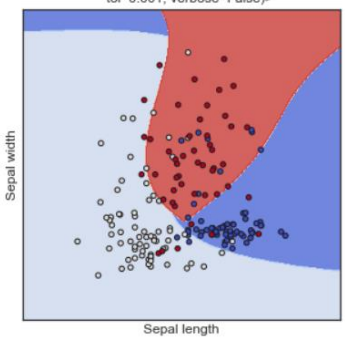
In [20]:
```
def make_meshgrid(x, y, h=.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    #Можно проверить все ли метки классов предсказываются
    #print(np.unique(Z))
    out = ax.contourf(xx, yy, Z, **params)
    return out
```

```
def plot_cl(clf):
    title = clf.__repr__
    clf.fit(wine_X, wine_y)
    fig, ax = plt.subplots(figsize=(5,5))
    X0, X1 = wine_X[:, 0], wine_X[:, 1]
    xx, yy = make_meshgrid(X0, X1)
    plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=wine_y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel('Sepal length')
    ax.set_ylabel('Sepal width')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
    plt.show()
```

In [21]:
```
wine_X_train, wine_X_test, wine_y_train, wine_y_test = train_test_split(
    wine.data, wine.target, test_size=0.2, random_state=1)
```

In [22]:
```
plot_cl(SVC(kernel='poly', degree=4, gamma=0.2, C=1.0))
```

```
<bound method BaseEstimator.__repr__ of SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=4, gamma=0.2, kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)>
```



In [23]:
```
svc = SVC(kernel='poly', degree=4, gamma=0.2, C=1.0).fit(wine_X_train, wine_y_train)
target_svc = svc.predict(wine_X_test)
```

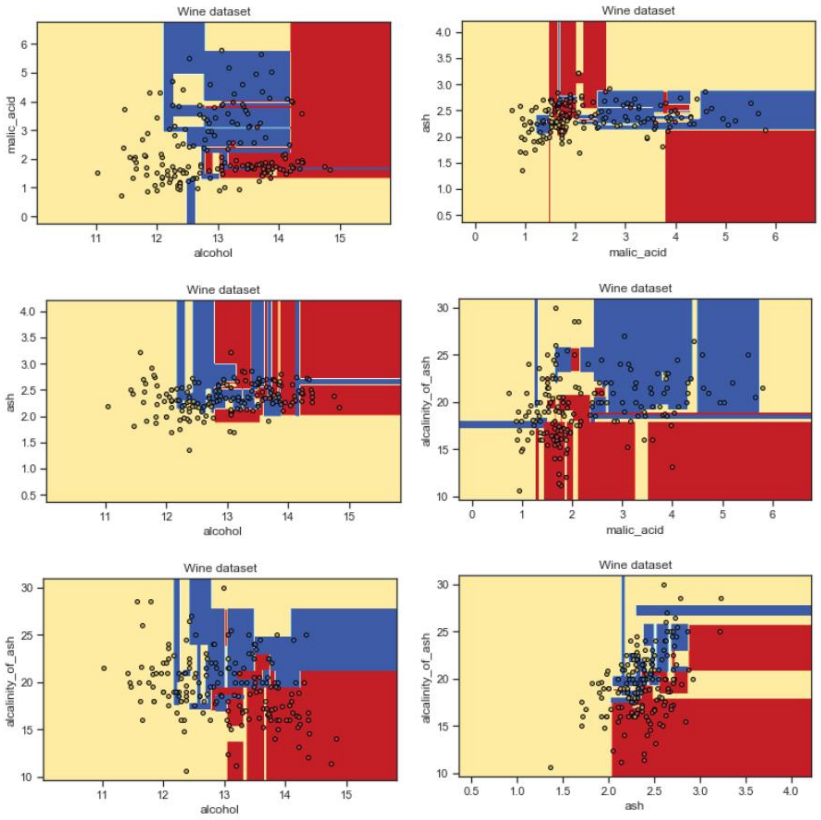
In [24]:
```
accuracy_score(wine_y_test, target_svc)
```

Out[24]: 0.9444444444444444

Построение модели «Дерево решений»:

In [25]:
```
def plot_tree_classification(title_param, ds):
    """
    Построение деревьев и вывод графиков для заданного датасета
    """

    n_classes = len(np.unique(ds.target))
    plot_colors = "ryb"
    plot_step = 0.02

    for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                    [1, 2], [1, 3], [2, 3]]):
        # We only take the two corresponding features
        X = ds.data[:, pair]
        y = ds.target

        # Train
        clf = DecisionTreeClassifier(random_state=1).fit(X, y)

        plt.title(title_param)

        x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
        y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                             np.arange(y_min, y_max, plot_step))
        plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)

        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

        plt.xlabel(ds.feature_names[pair[0]])
        plt.ylabel(ds.feature_names[pair[1]])

        # Plot the training points
        for i, color in zip(range(n_classes), plot_colors):
            idx = np.where(y == i)
            plt.scatter(X[idx, 0], X[idx, 1], c=color, label=ds.target_names[i],
                        cmap=plt.cm.RdYlBu, edgecolor='black', s=15)

        plt.show()
```

In [26]:
```
plot_tree_classification('Wine dataset', wine)
```



In [27]:
```
wine_x_ds = pd.DataFrame(data=wine['data'], columns=wine['feature_names'])
wine_x_ds.head()
```

Out[27]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od31
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	

In [28]:
```
# Обучим дерево на всех признаках wine
wine_tree_cl = DecisionTreeClassifier(random_state=1)
wine_tree_cl.fit(wine_x_ds, wine.target)
wine_tree_cl
```

Out[28]:
```
DecisionTreeClassifier(ccp_alpha=0.0, 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='deprecated',
                       random_state=1, splitter='best')
```

In [29]:
```
# Важность признаков
list(zip(wine_x_ds.columns.values, wine_tree_cl.feature_importances_))
```

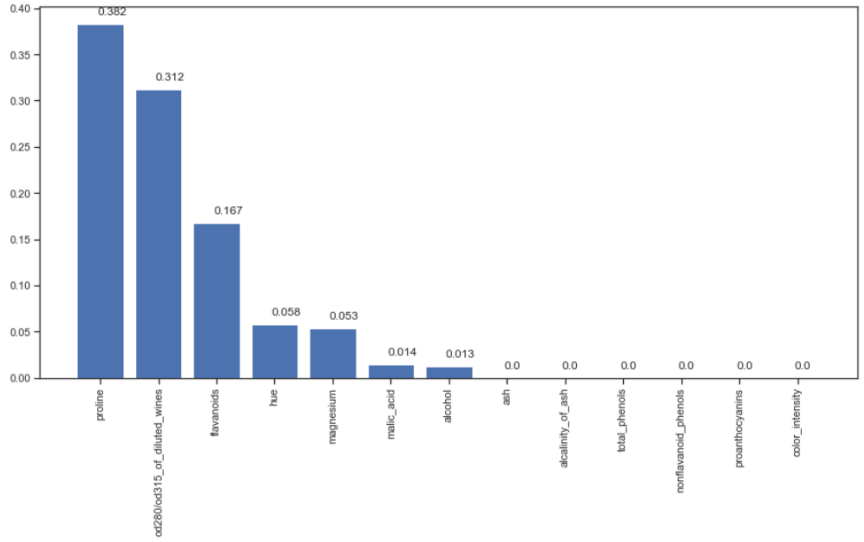
Out[29]:
```
[('alcohol', 0.012570564071187309),
 ('malic_acid', 0.014223159778821876),
 ('ash', 0.0),
 ('alcalinity_of_ash', 0.0),
 ('magnesium', 0.0534597951279922),
 ('total_phenols', 0.0),
 ('flavanoids', 0.16704836491408806),
 ('nonflavanoid_phenols', 0.0),
 ('proanthocyanins', 0.0),
 ('color_intensity', 0.0),
 ('hue', 0.05818509146040650 6),
 ('od280/od315_of_diluted_wines', 0.3120425747831769),
 ('proline', 0.38247044986432716)]
```

In [30]:
```
# Важность признаков в сумме дает единицу
sum(wine_tree_cl.feature_importances_)
```

Out[30]: 1.0

In [31]:
```
from operator import itemgetter

def draw_feature_importances(tree_model, X_dataset, figsize=(15,7)):
    """
    Вывод важности признаков в виде графика
    """
    # Сортировка значений важности признаков по убыванию
    list_to_sort = list(zip(X_dataset.columns.values, tree_model.feature_importances_))
    sorted_list = sorted(list_to_sort, key=itemgetter(1), reverse = True)
    # Названия признаков
    labels = [x for x,_ in sorted_list]
    # Важности признаков
    data = [x for _,x in sorted_list]
    # Вывод графика
    fig, ax = plt.subplots(figsize=figsize)
    ind = np.arange(len(labels))
    plt.bar(ind, data)
    plt.xticks(ind, labels, rotation='vertical')
    # Вывод значений
    for a,b in zip(ind, data):
        plt.text(a-0.05, b+0.01, str(round(b,3)))
    plt.show()
    return labels, data
```

In [32]:
```
wine_tree_cl_fl, wine_tree_cl_fd = draw_feature_importances(wine_tree_cl, wine_x_ds)
```



In [33]:
```
# Список признаков, отсортированный на основе важности, и значения важности
wine_tree_cl_fl, wine_tree_cl_fd
```

Out[33]:
```
(['proline',
  'od280/od315_of_diluted_wines',
  'flavanoids',
  'hue',
  'magnesium',
  'malic_acid',
  'alcohol',
  'ash',
  'alcalinity_of_ash',
  'total_phenols',
  'nonflavanoid_phenols',
  'proanthocyanins',
  'color_intensity'],
 [0.38247044986432716,
  0.3120425747831769,
  0.16704836491408806,
  0.058185091460406506,
  0.0534597951279922,
  0.014223159778821876,
  0.012570564071187309,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0])
```

In [34]:
```
wine_x_ds.head()
```

Out[34]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od31
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	

In [35]:
```
# Пересортируем признаки на основе важности
wine_x_ds_sorted = wine_x_ds[wine_tree_cl_fl]
wine_x_ds_sorted.head()
```

Out[35]:

	proline	od280/od315_of_diluted_wines	flavanoids	hue	magnesium	malic_acid	alcohol	ash	alcalinity_of_ash	total_phenols	nonflavanoid_phenols	proan
0	1065.0	3.92	3.06	1.04	127.0	1.71	14.23	2.43	15.6	2.80	0.28	
1	1050.0	3.40	2.76	1.05	100.0	1.78	13.20	2.14	11.2	2.65	0.26	
2	1185.0	3.17	3.24	1.03	101.0	2.36	13.16	2.67	18.6	2.80	0.30	
3	1480.0	3.45	3.49	0.86	113.0	1.95	14.37	2.50	16.8	3.85	0.24	
4	735.0	2.93	2.69	1.04	118.0	2.59	13.24	2.87	21.0	2.80	0.39	

In [36]:
```
# Разделим выборку на обучающую и тестовую
wine_X_train, wine_X_test, wine_y_train, wine_y_test = train_test_split(
    wine_x_ds_sorted, wine.target, test_size=0.5, random_state=1)
wine_X_train.shape, wine_X_test.shape
```

Out[36]:
```
((89, 13), (89, 13))
```

In [37]:
```
# Обучим дерево и предскажем результаты на всех признаках
wine_tree_cl_feat_1 = DecisionTreeClassifier(random_state=1).fit(wine_X_train, wine_y_train)
wine_y_test_predict = wine_tree_cl_feat_1.predict(wine_X_test)
wine_y_test_predict.shape
```

Out[37]:
```
(89,)
```

In [38]:
```
# Проверим точность по классам
print_accuracy_score_for_classes(wine_y_test, wine_y_test_predict)
```

```
Метка    Accuracy
0        0.9393939393939394
1        0.8529411764705882
2        0.9545454545454546
```

Таким образом, показатели метрики Accuracy говорят о высоком качестве всех построенных моделей.