Санкт-Петербургский Политехнический университет имени Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

ЛАБОРАТОРНАЯ РАБОТА №4

«АНСАМБЛЕВЫЕ МЕТОДЫ»

по дисциплине «Статистическое моделирование случайных процессов и систем»

Выполнил студент гр. 3530904/70103 Русаков Е.С.

Преподаватель Селин И.

Санкт-Петербург
2020

Оглавление

Задание

1. Исследуйте зависимость качества классификации от количества классификаторов в ансамбле для алгоритмов бэггинга на наборе данных glass.csv с различными базовыми классификаторами. Постройте графики зависимости качества классификации при различном числе классификаторов, объясните полученные результаты.
2. Исследуйте зависимость качества классификации от количества классификаторов в ансамбле для алгоритма бустинга (например, AdaBoost) на наборе данных vehicle.csv с различными базовыми классификаторами. Постройте графики зависимости качества классификации при различном числе классификаторов, объясните полученные результаты.
3. Постройте мета-классификатор для набора данных titanic.csv используя стекинг и оцените качество классификации.

Ход работы

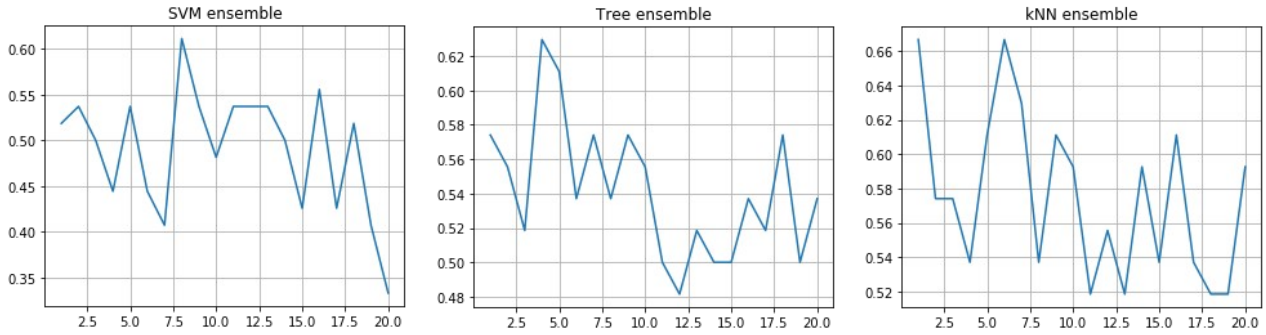
1. Bagging

Баггинг – методика построения сильного классификатора из некоторого числа однотианых слабых обычно путём формирования из оных «коллегии жюри».
В нашем эксперименте для компоновки ансамбля будет использоваться инструмент BaggingClassifier() из библиотеки Sklearn.ensemble.
Исследуем ансамбли на основе нескольких типов простых класификаторов: опорновекторного, деревьев решений и классификатора по ближайшим соседям.

а) Метод опорных векторов
Исходный тип слабого классификатора не будем объявлять слишком продвинутым. Остановимся на квадратичном ядре и gamma=1,0.
б) Деревья решений
Чтобы классификаторы не оказались слишком сложными, ограничим максимальную глубину, установив параметр расщепления равным 0.035.
в) Ближайшие соседи
Зададим параметр k=2. Больше в этом типе классификаторов настраивать нечего.

Из всего датасета glass.csv было выделено в самом начале 20% в качестве тестовой выборки, из остальной части выделялись пересекающиеся подвыборки для тренировки слабых классификаторов.

Как результат имеем следующие графики зависимости точности ансамбля от количества классификаторов в нём:



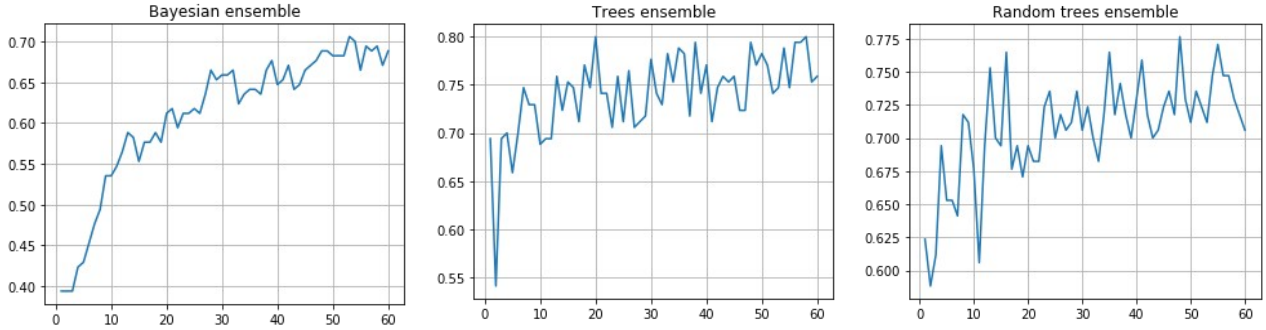
Вывод: по графикам видно, что при некотором размере «жюри» достигается максимум точности, после которого следует спад. Очевидно, получение полного эффекта прироста точности от баггинга зависит не только от числа слабых классификаторов, но и от типа классификаторов и объёма датасета, притом последний должен иметь размер пропорциональный размеру «жюри». В нашем же случае остаётся заключить, что построение ансамбля из классификаторов k ближайших соседей есть плохая идея.

2. Бустинг

Бустинг есть сродни баггингу, однако с некоторыми отличиями. Во-первых, обучение базовых классификаторов происходит на всей выборке целиком, а не на подвыборках. А во-вторых, каждый следующий малый классификатор обучается на основе ошибок предыдущего: веса неправильно распознанных примеров увеличиваются относительно распознанных верно, из-за чего классификатор «фокусируется» на слабых местах своего предшественника.

В нашем эксперименте будет использоваться метаклассификатор AdaBoostClassifier из библиотеки Sklearn. Как выяснилось опытным путём, не из каждого классификатора можно так построить ансамбль, поэтому в качестве базовых классификаторов будут использоваться только GaussianNB, ExtraTreeClassifier и DecisionTreeClassifier. Оба типа «ослаблены» установкой параметра расщепления равным 0,035.

Имеем следующие зависимости:



Вывод: ансамбль деревьев в целом достигает большей точности, однако делает это менее плавно, чем ансамбль байесовских классификаторов. Кроме того, случайные деревья оказались слишком случайными и ансамбль на их основе требует большее количество базовых классификаторов для достижения аналогичной точности.

3. Стэкинг

Стэкинг – метод построения ансамблей разнородных малых классификаторов, выход которых подаётся на вход классификатору(-ам) следующего слоя. Слоёв в общем случае может быть много. В этом задании мы построим простой двуслойный классификатор.

Датасет titaic.csv содержит данные о пассажирах затонувшего лайнера.
Прежде чем строить предсказатель для поля «Survived», нужно выделить существенные поля из массива данных.
Во-первых, избавимся от незначащих столбцов «PassengerId» и «Name», а также удалим данные о билетах и их ценах – столбцы «Ticket» и «Fare».
Во-вторых, обработаем столбец с местонахождением пассажиров. Заменим полные номера кают на обозначения блоков, например C85 → C, B115 → B, а пассажирам с палубы присвоим значение Z вместо NaN.
В-третьих, датасет насчитывает 176 человек, чей возраст неизвестен. Их придётся исключить из рассматриваемой выборки.
И наконец, требуется преобразовать метки в числовые значения, учтя их несравниваемость.

Поделим данные для обучения базовых моделей и метаклассификатора следующим образом:

<table>
<tr><td rowspan="3">Датасет после предобработки (100%)</td><td rowspan="2">Тренировочная выборка (80%)</td><td>48% для тренировки малых моделей</td><td>Случайные пересекающиеся выборки по 14,4% для каждой малой модели</td></tr>
<tr><td colspan="2">32% для тренировки метаклассификатора</td></tr>
<tr><td colspan="3">Тестовая выборка (20%)</td></tr>
</table>

Ансамбль будет иметь следующую архитектуру:

- В качестве базовых классификаторов будут выступать пять байесовских классификаторов и пять деревьев решений.
- Метаклассификатором будет трёхслойный персептрон с десятью входами, одним выходом и функцией активации tanh.

Результаты
На своих обучающих подвыборках базовые классификаторы достигали точностей 73% для байесовских и 93% для деревьев. Метаклассификатор обучался 50 эпох и достиг точности 82% на обучающей выборке.
На тестовой выборке ансамбль показал точность 83,2% со следующей матрицей ошибок:

72	10
14	47

Ради интереса сравним полученную точность с тем, как если бы мы на всей тестовой выборке тренировали один сильный классификатор вместо ансамбля.

Байесовский классификатор	Дерево решений (с ращеплением)

<table>
<tr><td>Точность: 76,9%</td><td>Точность: 82,5%</td></tr>
<tr><td>Матрица ошибок:<table><tr><td>59</td><td>23</td></tr><tr><td>10</td><td>51</td></tr></table></td><td>Матрица ошибок:<table><tr><td>74</td><td>8</td></tr><tr><td>17</td><td>44</td></tr></table></td></tr>
</table>

Используя один сильный классификатор, а именно дерево решений, мы бы получили точность сравнимую с точностью построенного ансамбля.

Приложение

Код программы на языке Python3

1. Баггинг

```
# задание 1. Баггинг
data = pd.read_csv('glass.csv').drop('Id', axis=1)
X_all = data.iloc[:, :-1].to_numpy()
Y_all = data.iloc[:, -1].to_numpy()

X, X_t, Y, Y_t = train_test_split(X_all, Y_all, test_size=0.2, shuffle=True)

model_svm = SVC(kernel='rbf', gamma=1.)
model_tree = DecisionTreeClassifier(min_samples_split=.035)
model_bayes = GaussianNB()
model_knn = KNeighborsClassifier(n_neighbors=2)

models = [ (model_svm, 'SVM'), (model_tree, 'Tree'), (model_knn, 'kNN') ]
numbers = [i for i in range(1, 21)]

for model, name in models:
    accs = []
    for classifiers_num in numbers:
        if (classifiers_num > 1):
            split_ratio = 1. / classifiers_num + 0.2
        else:
            split_ratio = 1.
        weights = [1. / split_ratio] * len(X)
        ensemble = BaggingClassifier(base_estimator=model,
n_estimators=classifiers_num,
                                     max_samples=split_ratio, n_jobs=4)
        ensemble.fit(X, Y)
        preds = ensemble.predict(X_t)
        accs.append(accuracy_score(preds, Y_t))

    fix, ax = plt.subplots(1, 1, figsize=(5,4))
    ax.plot(numbers, accs)
    ax.set_title(f'{name} ensemble')
    plt.grid(True)
    plt.show()
```

2. Бустинг

```
# Adaboost
data2 = pd.read_csv('vehicle.csv')
X_all = data2.iloc[:, :-1].to_numpy()
Y_all = LabelEncoder().fit_transform(data2.iloc[:, -1].to_numpy())

X, X_t, Y, Y_t = train_test_split(X_all, Y_all, test_size=0.2, shuffle=True)

model_tree = DecisionTreeClassifier(min_samples_split=.035)
model_ex_tree = ExtraTreeClassifier(min_samples_split=.035)
model_bayes = GaussianNB()
model_svm = SVC(kernel='rbf', gamma=1.)

models = [ (model_bayes, 'Bayesian'), (model_svm, 'SVM'), (model_tree,
'Trees'), (model_ex_tree, 'Random trees') ]
numbers = [i for i in range(1, 101)]
```

```
for model, name in models:
    accs = []
    for classifiers_num in numbers:
        if (classifiers_num > 1):
            split_ratio = 1. / classifiers_num + 0.2
        else:
            split_ratio = 1.
        weights = [1. / split_ratio] * len(X)
        ensemble = AdaBoostClassifier(base_estimator=model,
n_estimators=classifiers_num, algorithm='SAMME')
        ensemble.fit(X, Y)
        #preds = ensemble.predict(X_t)
        #accs.append(accuracy_score(preds, Y_t))
        accs.append(ensemble.score(X_t, Y_t))

    fix, ax = plt.subplots(1, 1, figsize=(5,4))
    ax.plot(numbers, accs)
    ax.set_title(f'{name} ensemble')
    plt.grid(True)
    plt.show()
```

3. Стэкинг

```
# Стацкинг

data3 = pd.read_csv('titanic.csv').drop(['PassengerId', 'Name', 'Ticket',
'Fare'], axis=1)
data3['Cabin'].fillna('Z', inplace=True)
data3.dropna(axis=0, inplace=True)
data3.index = [i for i in range(len(data3))]


for i in range(len(data3)):
    data3.at[i, 'Cabin'] = data3.loc[i, 'Cabin'][0]
    if (data3.at[i, 'Sex'] == 'male'):
        data3.at[i, 'Sex'] = 1
    else:
        data3.at[i, 'Sex'] = 0

data3 = pd.get_dummies(data3, columns=['Cabin', 'Embarked'])

X_all = data3.iloc[:, 1:].to_numpy()
Y_all = data3.iloc[:, 0].to_numpy()

X, X_test, Y, Y_test = train_test_split(X_all, Y_all, test_size=0.2,
shuffle=True)
X_basic, X_meta, Y_basic, Y_meta = train_test_split(X, Y, test_size=0.4,
shuffle=True)

from keras.models import Sequential
from keras.layers import Dense


small_models = [  GaussianNB(),
                  GaussianNB(),
```

```
                  GaussianNB(),
                  GaussianNB(),
                  GaussianNB(),
                  DecisionTreeClassifier(min_samples_split=.035),
                  DecisionTreeClassifier(min_samples_split=.035),
                  DecisionTreeClassifier(min_samples_split=.035),
                  DecisionTreeClassifier(min_samples_split=.035),
                  DecisionTreeClassifier(min_samples_split=.035) ]

final_model = Sequential([ Dense(10, input_dim=10, activation='tanh'),
                           Dense(5, activation='tanh'),
                           Dense(1, activation='tanh') ])
final_model.compile(loss='mse', optimizer='rmsprop', metrics=['accuracy'])

# small models training
for model in small_models:
    small_X, _, small_Y, _ = train_test_split(X_basic, Y_basic, test_size=0.7,
shuffle=True)
    model.fit(small_X, small_Y)
    print(model.score(small_X, small_Y))

# meta-model training
preds = small_models[0].predict(X_meta).reshape(len(X_meta),1)

for model in small_models[1:]:
    preds = np.concatenate((preds,
model.predict(X_meta).reshape(len(X_meta),1)), axis=1)

final_model.fit(preds, Y_meta, epochs=50)

# overall ensamble evaluating
def ensamble_predict(small_models, final_model, X_data):
    preds = small_models[0].predict(X_data).reshape(len(X_data),1)

    for model in small_models[1:]:
        preds = np.concatenate((preds,
model.predict(X_data).reshape(len(X_data),1)), axis=1)

    final_preds = final_model.predict(preds).reshape((len(X_data)))
    return np.round(final_preds).astype('int')

Y_preds = ensamble_predict(small_models, final_model, X_test)

conf_matr = pd.DataFrame(confusion_matrix(Y_test, Y_preds))
print(f'Точность ансамбля: {accuracy_score(Y_test, Y_preds):.3f}')
conf_matr

sample_model = GaussianNB()
sample_model.fit(X, Y)
Y_preds2 = sample_model.predict(X_test)
conf_matr = pd.DataFrame(confusion_matrix(Y_test, Y_preds2))
print(f'Точность байесовского классификатора: {accuracy_score(Y_test,
Y_preds2):.3f}')
conf_matr

sample_model = DecisionTreeClassifier(min_samples_split=.035)
sample_model.fit(X, Y)
```

```
Y_preds2 = sample_model.predict(X_test)
conf_matr = pd.DataFrame(confusion_matrix(Y_test, Y_preds2))
print(f'Точность дерева: {accuracy_score(Y_test, Y_preds2):.3f}')
conf_matr
```