Считываем данные, делаем выборку параметров, дамми-переменные и классификация методом опорных векторов(SVM)

```
#Импорт модулей
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.svm import SVC
from sklearn.metrics import precision_score, recall_score, f1_score

# Загрузка данных
exam_df = pd.read_csv('StudentsPerformance.csv')

#Выберем необходимые нам параметры
exam_df = exam_df.loc[:, exam_df.columns.isin(['gender', 'parental
level of education', 'writing score', 'reading score', 'math
score',])]

#Смотрим на наши данные
print(exam_df)

#Удаление пустых значений (если таковы имеются)
exam_df = exam_df.dropna()

# Выделение целевого признака ( writing score (выше среднего значения
– класс 0, ниже или совпадает – класс 1) )
exam_df['writing_score_above_avg'] = (exam_df['writing score'] >
exam_df['writing score'].mean()).astype(int)

# Удаление целевого признака
exam_df.drop('writing score', axis=1, inplace=True)

# Создание дамми-переменных с помощью one-hot-encoding

#Пол (Мужчина - 1, женщина - 0)
exam_df['gender'] = np.where(exam_df['gender'] == 'male' , 0, 1)

#Уровень образования (Бакалавр - 1, Магистр - 2, Аспирант - 3, High
school - 4, Колледж - 5, Some high school - 6)
exam_df['parental level of education'] = np.where(exam_df['parental
level of education'] == 'bachelor\'s degree' ,
                                                  1, exam_df['parental
level of education'])
exam_df['parental level of education'] = np.where(exam_df['parental
level of education'] == 'master\'s degree' ,
                                                  2, exam_df['parental
level of education'])
exam_df['parental level of education'] = np.where(exam_df['parental
level of education'] == 'associate\'s degree' ,
```

```
                                                    3, exam_df['parental
level of education'])
exam_df['parental level of education'] = np.where(exam_df['parental
level of education'] == 'high school' ,
                                                    4, exam_df['parental
level of education'])
exam_df['parental level of education'] = np.where(exam_df['parental
level of education'] == 'some college' ,
                                                    5, exam_df['parental
level of education'])
exam_df['parental level of education'] = np.where(exam_df['parental
level of education'] == 'some high school' ,
                                                    6, exam_df['parental
level of education'])

df_encoded = exam_df

# Разделение на обучающую и тестовую выборки
X_training, X_testing, y_training, y_testing =
train_test_split(df_encoded.drop('writing_score_above_avg', axis=1),

df_encoded['writing_score_above_avg'],
                                                      test_size = 0.3,
                                                      random_state = 42)

# Создание и обучение модели SVM
clf = SVC(kernel='linear', random_state=42)
clf.fit(X_training, y_training)

# Получение прогнозов на тестовой выборке
y_pred = clf.predict(X_testing)

# Оценка точности классификатора с помощью метрик precision, recall и
F1
precision = precision_score(y_testing, y_pred)
recall = recall_score(y_testing, y_pred)
f1 = f1_score(y_testing, y_pred)
```

```
     gender parental level of education  math score  reading score
0         1                            1          72             72  \
1         1                            5          69             90
2         1                            2          90             95
3         0                            3          47             57
4         0                            5          76             78
..      ...                          ...         ...            ...
995       1                            2          88             99
996       0                            4          62             55
997       1                            4          59             71
998       1                            5          68             78
999       1                            5          77             86
```

```
     writing_score_above_avg
0                          1
1                          1
2                          1
3                          0
4                          1
..                       ...
995                        1
996                        0
997                        0
998                        1
999                        1

[1000 rows x 5 columns]
```

Итог SVM

```
#Результат
print( "accuracy:"+str(np.average(cross_val_score(clf, X_testing,
y_testing, scoring=  'accuracy'))))
print(       "f1:"+str(np.average(cross_val_score(clf, X_testing,
y_testing, scoring=  'f1'))))
print("precision:"+str(np.average(cross_val_score(clf, X_testing,
y_testing, scoring=  'precision'))))
print(   "recall:"+str(np.average(cross_val_score(clf, X_testing,
y_testing, scoring=  'recall'))))
```

```
accuracy:0.9066666666666668
f1:0.9063845372228225
precision:0.9185974945533768
recall:0.9
```

Классификация с помощью классификатора типа Случайный лес(Random forest)

```
#Подключаем модуль
from sklearn.ensemble import RandomForestClassifier

# разделение данных на обучающую и тестовую выборки
training_data, testing_data, training_labels, testing_labels =
train_test_split(
    df_encoded.drop("writing_score_above_avg", axis=1),
df_encoded["writing_score_above_avg"], test_size=0.2, random_state=42)

# построение классификатора
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(training_data, training_labels)

# оценка качества классификатора на тестовой выборке
```

```
print( "accuracy:"+str(np.average(cross_val_score(rf, X_testing,
y_testing, scoring=  'accuracy'))))
print(       "f1:"+str(np.average(cross_val_score(rf, X_testing,
y_testing, scoring=  'f1'))))
print("precision:"+str(np.average(cross_val_score(rf, X_testing,
y_testing, scoring=  'precision'))))
print(   "recall:"+str(np.average(cross_val_score(rf, X_testing,
y_testing, scoring=  'recall'))))
```

```
accuracy:0.8966666666666667
f1:0.896499688318275
precision:0.9037908029843514
recall:0.8933333333333333
```

Перебираем различные гиперпараметры для Random Forest с помощью GridSearch

```
#Делаем импорт ещё пары модулей необходимых нам
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV

# определение диапазона значений для перебора
param_grid = {
    "n_estimators": [50, 100, 150, 200],
    "max_depth": [None, 5, 10, 15],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "max_features": ["sqrt", "log2"],
}

# создание экземпляра класса GridSearchCV
grid_search_rf = GridSearchCV(RandomForestClassifier(random_state=42),
param_grid, cv=3, n_jobs=-1)

# запуск GridSearchCV
grid_search_rf.fit(training_data, training_labels)

# Оцениваем качество классификации
print(classification_report(y_testing, y_pred))

print( "accuracy:"+str(np.average(cross_val_score(grid_search_rf,
X_testing, y_testing, scoring=  'accuracy'))))
print(       "f1:"+str(np.average(cross_val_score(grid_search_rf,
X_testing, y_testing, scoring=  'f1'))))
print("precision:"+str(np.average(cross_val_score(grid_search_rf,
X_testing, y_testing, scoring=  'precision'))))
print(   "recall:"+str(np.average(cross_val_score(grid_search_rf,
X_testing, y_testing, scoring=  'recall'))))
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91       150
           1       0.94      0.88      0.91       150

    accuracy                           0.91       300
   macro avg       0.91      0.91      0.91       300
weighted avg       0.91      0.91      0.91       300

accuracy:0.8933333333333333
f1:0.8939327871944979
precision:0.8978257032692516
recall:0.8933333333333333
```

Лучшими гиперпараметрами оказались:

```
print("Best parameters:", grid_search_rf.best_params_)
```

```
Best parameters: {'max_depth': 5, 'max_features': 'sqrt',
'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 150}
```

Различные комбинации гиперпараметров для случайного дерева с шагом 50 и 10 в параметре n_estimators и вывод их лучших значений

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, recall_score, f1_score

# Различные комбинации гиперпараметров для случайного дерева с шагом
50 в параметре n_estimators
param_grid_50 = {
    'n_estimators': [i for i in range(150, 301, 50)]
}

rfc_50 = RandomForestClassifier(random_state=42)
grid_search_50 = GridSearchCV(estimator=rfc_50,
param_grid=param_grid_50, cv=5)
grid_search_50.fit(X_training, y_training)

# Вывод лучших значений для шага 50 в параметре n_estimators
print("Best n_estimators for step 50: ",
grid_search_50.best_params_['n_estimators'])

# Прогноз на основе данных тестирования с помощью случайного дерева с
шагом 50 в параметре n_estimators
rfc_50_pred = grid_search_50.predict(X_testing)

# Точность, отзыв и оценку F1, используя метод случайный лес с шагом
50
rfc_50_accuracy = accuracy_score(y_testing, rfc_50_pred)
```

```
rfc_50_recall = recall_score(y_testing, rfc_50_pred,
average='weighted')
rfc_50_f1_score = f1_score(y_testing, rfc_50_pred, average='weighted')

print("Accuracy for step 50: ", rfc_50_accuracy)
print("Recall for step 50: ", rfc_50_recall)
print("F1 Score for step 50: ", rfc_50_f1_score)

# Различные комбинации гиперпараметров для случайного дерева с шагом
10 в параметре n_estimators
param_grid_10 = {
    'n_estimators': [i for i in range(150, 301, 10)]
}

rfc_10 = RandomForestClassifier(random_state=42)
grid_search_10 = GridSearchCV(estimator=rfc_10,
param_grid=param_grid_10, cv=5)
grid_search_10.fit(X_training, y_training)

# Вывод лучших значений для шага 10 в параметре n_estimators
print("Best n_estimators for step 10: ",
grid_search_10.best_params_['n_estimators'])

# Прогноз на основе данных тестирования с помощью случайного дерева с
шагом 10 в параметре n_estimators
rfc_10_pred = grid_search_10.predict(X_testing)

# Точность, отзыв и оценку F1, используя метод случайный лес с шагом
10
rfc_10_accuracy = accuracy_score(y_testing, rfc_10_pred)
rfc_10_recall = recall_score(y_testing, rfc_10_pred,
average='weighted')
rfc_10_f1_score = f1_score(y_testing, rfc_10_pred, average='weighted')

print("Accuracy for step 10: ", rfc_10_accuracy)
print("Recall for step 10: ", rfc_10_recall)
print("F1 Score for step 10: ", rfc_10_f1_score)
```

```
Best n_estimators for step 50:  200
Accuracy for step 50:  0.9
Recall for step 50:  0.9
F1 Score for step 50:  0.899888765294772
Best n_estimators for step 10:  200
Accuracy for step 10:  0.9
Recall for step 10:  0.9
F1 Score for step 10:  0.899888765294772
```

В итоге классификатор типа SVM справился чуть лучше(и быстрее)