Model Training and Improvement

За статиите - Две доказано добри научни статии и да ги докарам във вид, който е лесен за разбиране дори от човек със слаба представа от тази материя!!!! Peer review, репродуциране на статиите. Резюме на статията (Jupyter notebook)

1. Train-test split

Train-test split (разделяне на тренировъчни и тестови данни) е основна практика в машинното обучение, която включва разделяне на наличния набор от данни на две отделни части:

1. **Тренировъчен набор (Train set):**

- Използва се за обучаване на модела.
- Моделът "учи" от тези данни, за да открие модели, зависимости и връзки между входните характеристики и целевата променлива.

2. **Тестов набор (Test set):**

- Използва се за оценка на представянето на вече обучен модел.
- Данните в този набор не се използват по време на обучението, което позволява обективна оценка на способността на модела да се справя с нови, невиждани данни.

Защо е важно да се прави train-test split:

- **Предотвратяване на Overfitting:** Ако моделът бъде обучен и тестван върху едни и същи данни, той може да научи специфичните характеристики на тези данни, вместо да обобщава общите модели. Това води до лошо представяне при работа с нови данни.
- **Обективна оценка:** Разделянето осигурява независим набор от данни за тестване, което позволява реална оценка на ефективността на модела.

2. pd.get_dummies()

`pandas.get_dummies()` е функция в библиотеката `pandas`, която се използва за преобразуване на категорийни променливи в числови индикаторни (dummy) променливи. Това е важна стъпка в предварителната обработка на данни, особено когато работите с

машинно обучение, тъй като повечето алгоритми изискват числови входни данни. dataset-а трябва да бъде само от числа, когато го подаваме на sklearn

2.1 Какво прави `get_dummies`:

- Преобразува категорийни (неколичествени) данни в бинарни (0 или 1) колони.
- Създава нови колони за всяка уникална категория в оригиналната колона.
- Това позволява на моделите да обработват категорийни данни като числови входове.

2.2 Пример с код:

```
import pandas as pd

# Примерни данни
data = {
    'Продукт': ['Ябълка', 'Банан', 'Ябълка', 'Киви', 'Банан'],
    'Цвят': ['Червен', 'Жълт', 'Червен', 'Зелен', 'Жълт'],
    'Количество': [10, 15, 10, 5, 15]
}

df = pd.DataFrame(data)
df
```

Оригинален DataFrame:

	Продукт	Цвят	Количество
0	Ябълка	Червен	10
1	Банан	Жълт	15
2	Ябълка	Червен	10
3	Киви	Зелен	5
4	Банан	Жълт	15

Използване на `pd.get_dummies`:
Нека преобразуваме колоната `Цвят` в dummy променливи:

```
# Преобразуване на категорийната колона 'Цвят' в dummy променливи
df_dummies = pd.get_dummies(df, columns=['Цвят'])
df_dummies
```

DataFrame след прилагане на get_dummies:

	Продукт	Количество	Цвят_Жълт	Цвят_Зелен	Цвят_Червен
0	Ябълка	10	False	False	True
1	Банан	15	True	False	False
2	Ябълка	10	False	False	True
3	Киви	5	False	True	False
4	Банан	15	True	False	False

Обяснение:

1. **Оригиналният DataFrame** съдържа категорийна колона `Цвят` с три уникални стойности: `'Червен'`, `'Жълт'` и `'Зелен'`.
2. `pd.get_dummies` преобразува тази колона в три нови бинарни колони:
   - `Цвят_Червен`: 1 ако цвятът е червен, иначе 0.
   - `Цвят_Жълт`: 1 ако цвятът е жълт, иначе 0.
   - `Цвят_Зелен`: 1 ако цвятът е зелен, иначе 0.
3. **Останалите колони** (`Продукт` и `Количество`) остават непроменени, освен ако не бъдат специфицирани за преобразуване.

2.3 Допълнителни опции:

1. **Дропване на първата категория (`drop_first=True`):** Това предотвратява проблема с мултиколинейността, като премахва една от dummy променливите. Например, ако `drop_first=True`, само две от трите категории ще бъдат представени с dummy променливи.

```
df_dummies_drop = pd.get_dummies(df, columns=['Цвят'], drop_first=True)
df_dummies_drop
```

DataFrame с drop_first=True:

	Продукт	Количество	Цвят_Зелен	Цвят_Червен
0	Ябълка	10	False	True
1	Банан	15	False	False
2	Ябълка	10	False	True
3	Киви	5	True	False
4	Банан	15	False	False

В този случай, ако и двете dummy променливи са 0, това означава, че цветът е първата категория (`'Жълт'`).

3. OneHotEncoder()

`OneHotEncoder()` е инструмент от библиотеката scikit-learn, използван за преобразуване на категориални (номинални) характеристики в числов формат. Той преобразува всяка уникална категория в отделен бинарен (0 или 1) стълб. Това е особено полезно за алгоритми за машинно обучение, които работят с числови данни и не могат директно да обработват категориални характеристики.

4. OneHotEncoder() vs get_dummies()

Основни разлики между `OneHotEncoder` и `pd.get_dummies`:

4.1 Интеграция с машинно обучение (ML) пайплайни

- **OneHotEncoder**:
  - Част от **scikit-learn**, което го прави лесно интегрируем в ML пайплайни.
  - Може да бъде използван заедно с други трансформери и модели в scikit-learn, което улеснява автоматизацията на процеса на обработка на данни и обучение на модели.
- `pd.get_dummies`:
  - Функция на **pandas**, която е по-подходяща за бързо преобразуване на данни преди анализ или визуализация.

- Не е директно интегриран в scikit-learn пайплайни, което може да изисква допълнителни стъпки за включване в ML процеси.

4.2 Обработка на неизвестни категории

- **OneHotEncoder**:
  - Поддържа параметъра `handle_unknown`, който позволява обработка на категории, които не са били срещани по време на обучението (например, игнориране или присвояване на специфична стойност).
  - Това е особено полезно при трансформиране на нови данни, съдържащи непознати категории.
- `pd.get_dummies`:
  - Не поддържа директно обработка на неизвестни категории при трансформиране на нови данни.
  - Ако новите данни съдържат категории, които не са били присъстващи в оригиналния DataFrame, те няма да бъдат обработени правилно без допълнителни стъпки.

4.3 Изходен формат

- **OneHotEncoder**:
  - Може да върне разредена матрица (sparse matrix), което е по-ефективно по отношение на паметта при големи набори от данни с много категории.
  - Подходящо за модели, които могат да работят директно с разредени матрици (например, логистична регресия, линейни модели).
- `pd.get_dummies`:
  - Винаги връща плосък (dense) DataFrame, което може да заема повече памет при големи набори от данни.
  - Подходящо за по-малки до средни набори от данни и за ситуации, когато разредените матрици не са необходими.

-"sparse matrix" е вид матрица, в която повечето елементи са нули или липсващи стойности. Това е противоположността на плътната матрица (dense matrix), където повечето елементи са различни от нула.

4.4 Производителност и мащабируемост

- **OneHotEncoder**:

    - По-подходящ за големи набори от данни, особено когато се използват разредени матрици.
    - По-ефективен при обработка на данни в рамките на ML пайплайни.
- `pd.get_dummies`:
    - Подходящ за по-малки до средни набори от данни.
    - Може да бъде по-бавен и по-малко ефективен при много големи или сложни набори от данни.

4.5. Заключение

Изборът между **OneHotEncoder** и `pd.get_dummies` зависи от конкретните нужди на вашия проект:

- **Изберете OneHotEncoder**, ако работите върху машинно обучение проекти, които изискват интеграция в scikit-learn пайплайни, обработка на неизвестни категории, или ако работите с големи и сложни набори от данни.
- **Изберете** `pd.get_dummies`, ако имате нужда от бързо и лесно преобразуване на категориални данни за анализ, визуализация или за подготовка на данни, които няма да се използват директно в scikit-learn модели.

5. Pipeline

`Data Pipeline` (поток от данни) в контекста на машинното обучение (ML) представлява последователност от стъпки или процеси, които обработват и трансформират данните от източника им до финалния модел за машинно обучение. Целта на data pipeline е да осигури ефективно, надеждно и мащабируемо обработване на данните, което да поддържа създаването и поддържането на ML модели.

5.1 Защо Data Pipeline е важен в ML?

- **Ефективност и автоматизация:** Автоматизираните потоци от данни позволяват бързо и последователно обработване на големи обеми данни.
- **Надеждност и проследяемост:** Осигурява възпроизводимост на процесите и лесно проследяване на източниците и трансформациите на данните.
- **Мащабируемост:** Позволява обработка на нарастващи обеми данни без загуба на производителност.
- **Качество на данните:** Подобрява качеството на данните, което е критично за точността и ефективността на ML моделите.

- **Бързо разгръщане на модели:** Позволява по-бързо обновяване и интеграция на нови модели в продукционната среда.

!!! Добре е в края на pipelin-а да имаме `estimator - (оценител)` !!!

5.2 `pipeline.steps`

Атрибут в `scikit-learn`, който описва стъпките в **Pipeline** обект. Всяка стъпка представлява трансформер или оценител, който изпълнява специфична функция в процеса на машинно обучение.

Структура

- **Формат:** `pipeline.steps` е списък от кортежи, всеки от които съдържа:
  - **Име на стъпката** (стринг)
  - **Трансформер или оценител** (обект, реализиращ методите `fit`, `transform` и/или `fit_predict`)

6.ColumnTransformer()

6.1 Определение

`ColumnTransformer` е мощен инструмент в библиотеката scikit-learn, който позволява прилагането на различни трансформации върху различни колони (характеристики) на вашия набор от данни. Това е особено полезно, когато работите с хетерогенни данни, съдържащи както числови, така и категориални променливи.

С `ColumnTransformer` можем да създадем предварителни обработващи стъпки (preprocessing steps) за различните типове данни в единен pipeline, което улеснява целия процес на моделиране и подобрява повторяемостта на кода.

6.2 Параметърът `remainder`

Определя как да се обработват колоните, които **не са изрично посочени** в трансформерите. Това е полезно, когато искате да прилагате трансформации само на определени колони, докато останалите да бъдат оставени непроменени или обработени по друг начин.

Възможни стойности за `remainder`:

1. `'drop'` (по подразбиране): Оставените колони се игнорират и **не** се включват в резултатния набор от данни.
2. `'passthrough'`: Оставените колони се оставят **непроменени** и се включват в резултатния набор от данни.
3. **Всякакъв друг трансформър**: Можете да зададете собствен трансформър за обработка на оставените колони.

7. FunctionTransformer()

7.1 Определение

`FunctionTransformer` е трансформатор от библиотеката **scikit-learn**, който позволява прилагането на потребителски функции към данни вътре в конвейери (pipelines) за обработка на данни. Това е особено полезно, когато е необходимо да се интегрират собствени преобразувания на данни в стандартния процес на машинно обучение.

7.2 Пример за използване на `FunctionTransformer`

Нека разгледаме пример, в който искаме да приложим логаритмично преобразуване към числови данни преди обучението на модел. Това може да бъде полезно, например, за обработка на характеристики с експоненциално разпределение.

```
def log_transform(X):
        return np.log(X + 1) # Добавяме 1, за да избегнем log(0)

log_transformer = FunctionTransformer(log_transform, validate=True)

pipeline = Pipeline([
        ('log_transform', log_transformer),
        ('linear_regression', LinearRegression())
])
```

8. Запазване на модел във файл - pickle

8.1 Определение

Модулът `pickle` в Python позволява *сериализиране* и десериализиране на обекти, което е полезно за запазване на модели или данни във файл, както и за последващото им възстановяване. Това е подходящо, например, когато искате да запазите обучен модел и след това да го заредите отново без нужда от повторно обучение.

Сериализация е процесът на преобразуване на обект (например променлива, структура от данни или модел) в формат, който може да бъде съхранен (например във файл) или предаден през мрежа, а след това възстановен обратно до оригиналната си форма.

8.2 pickle.dump()

`pickle.dump(object, file)` — сериализира обекта и го записва във файл.

```
import pickle

# Примерен обучен модел (например от sklearn)
model = ...  # Това е моят модел

# Запазване на модела във файл
with open('model.pkl', 'wb') as file:
    pickle.dump(model, file)
```

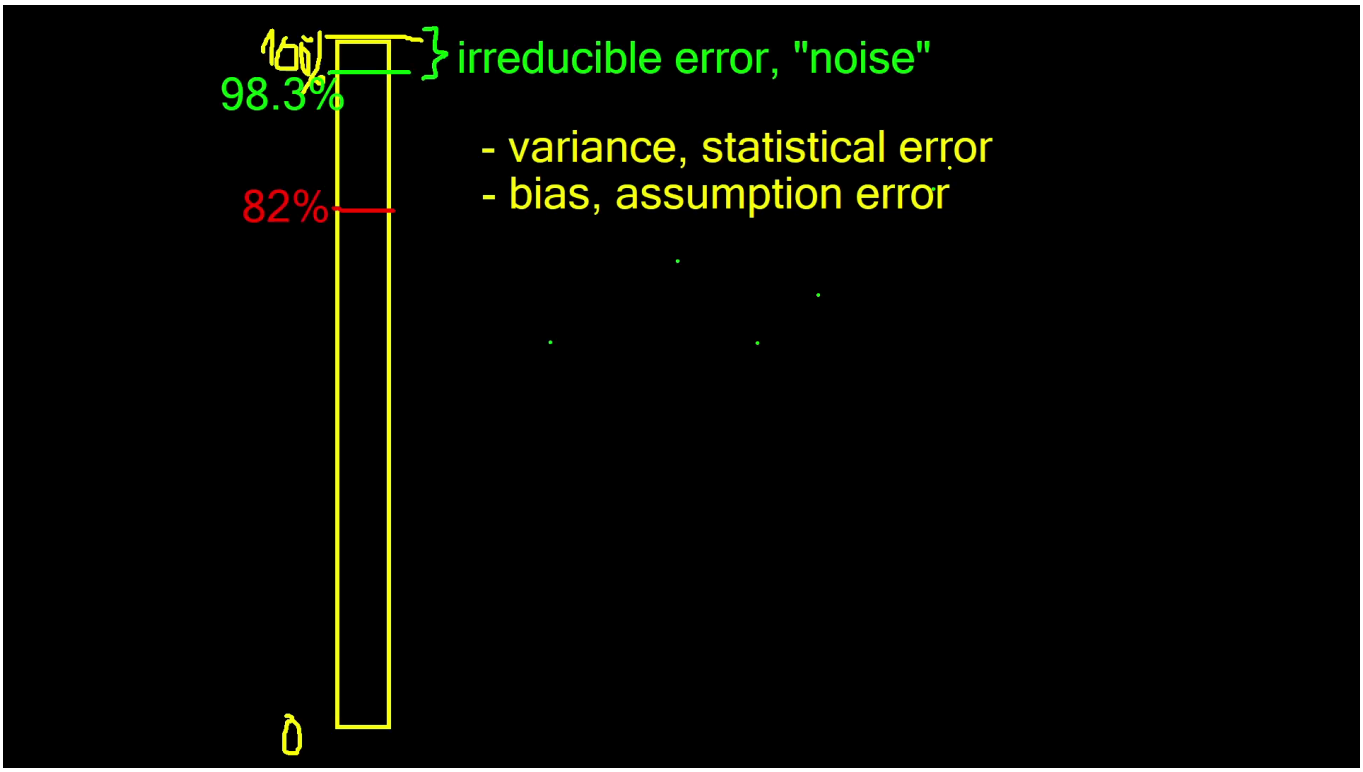
8.3 pickle.load()

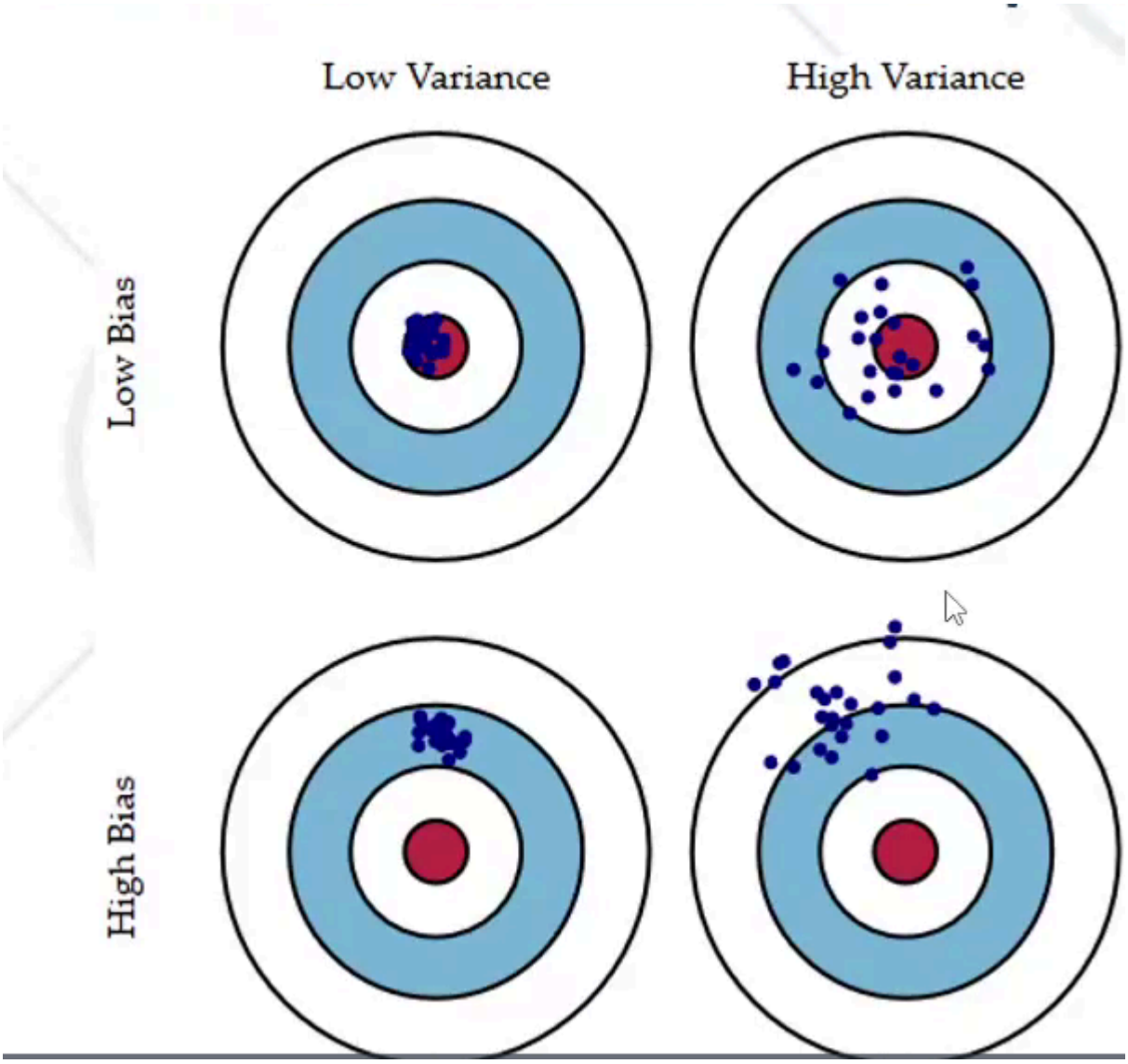
`pickle.load(file)` — десериализира обект от файл и го връща в Python.

```
import pickle

# Зареждане на модела от файл
with open('model.pkl', 'rb') as file:
    loaded_model = pickle.load(file)

# Сега мога да използвам заредения модел
```


100%
98.3%
82%
0
irreducible error, "noise"
- variance, statistical error
- bias, assumption error


Low Variance
High Variance
Low Bias
High Bias

Regularization

- Logistic regression C param
  Mean Squared error (MSE)

One of the most important rules in machine learning is

- **NEVER test the model with the data you trained it on!**

train_test_split
stratify?

```
attributes_train, attributes_test, target_train, target_test = train_test_split(attributes, target)
```

metrics
ROC curve

Cross-validation
test set and validation set
GridSerachCV