KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 7 урок > СТАТИСТИКА

> Оглавление

> Предсказание значений

После того, как мы оценили взаимосвязь между переменными и вычислили b_0 и b_1 , мы можем пойти дальше: предсказывать значения зависимой переменной,

подставляя значения независимой в нашу модель.

$$Y = b_0 + b_1(\text{значениеНП}) + \epsilon$$

где ϵ - ошибка между предсказаниями модели и реальными значениями.

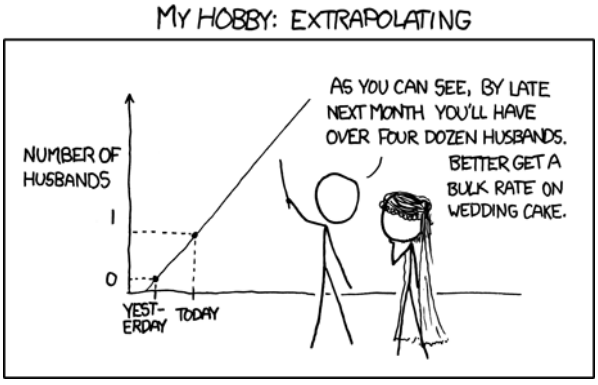
Фактически линейная регрессия – это простейший пример машинного обучения, и на практике вам с высокой вероятностью придётся строить подобные предсказательные модели.

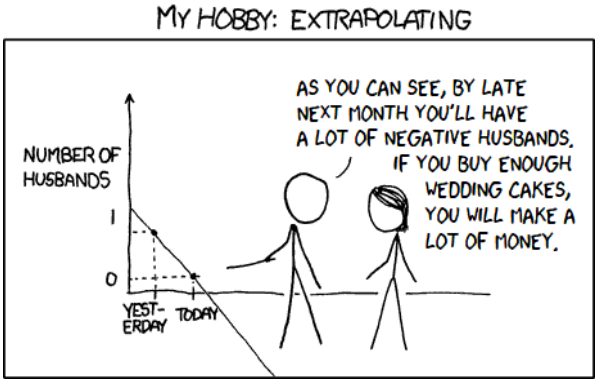
О чём стоит помнить:

- Для качественных предсказаний нужна качественная модель – убедитесь, что модель соответствует её допущениям
- Ошибка экстраполяции: если данные в нашей выборке показывают линейный тренд, это не означает, что за их пределами тренд останется линейным
- Вдогонку к предыдущему: убедитесь, что предсказанные значения подходят с точки зрения здравого смысла
- Как и всегда, сам по себе факт того, что мы можем построить такую модель и предсказывать по ней значения, не говорит о причинно-следственных взаимосвязях между переменными

```
# на примере statsmodels
# X - массив со значениями НП, model - построенная регрессионная модель

model.predict(X)
```





> Множественные предикторы (НП)

На практике чаще всего у нас есть несколько НП, и мы ходим предсказывать ЗП по их сочетанию. В таком случае коэффициентов угла наклона у нас становится больше одного (но b_0 остается один), и основное уравнение регрессии начинает выглядеть так:

$$Y = b_0 + b_1X_1 + b_2X_2 + ... + b_nX_n + \epsilon$$

где n – это номер предиктора и соответствующего ему коэффициента.

Условия применения:

1. Линейная зависимость переменных
2. Остатки нормально распределены
3. Переменные нормально распределены (если верен пункт 2, то необязательно)
4. Проверка на гетероскедастичность (различия в дисперсии на разных уровнях НП, дисперсия должна быть примерно одинакова)
5. **Проверка на мультиколлинеарность**

Мультиколлинеарность – феномен, возникающий при сильной корреляции между независимыми переменными. При мультиколлинеарности стандартные ошибки коэффициентов линейной регрессии искусственно завышаются, что приводит к повышенной ошибке II рода. Если убрать высококоррелирующую переменную из числа НП, то это может привести к увеличению качества модели, а коэффициенты будут значимы.

Внимание: мультиколлинеарность не влияет на сами значения коэффициентов регрессии – только на их стандартные ошибки. По этой причине модель всё ещё может иметь хорошую предсказательную силу.

Для оценки коллинеарности между переменными используется т.н. **фактор инфляции дисперсии** *(variance inflation factor, VIF)*. Считается, что значения VIF больше пяти могут означать высокую коллинеарность данной переменной с

остальными. Однако стоит помнить, что это не означает необходимость удалять эту переменную из модели – это решение стоит принимать, исходя из особенностей задачи. Пример VIF в контексте Python: https://etav.github.io/python/vif_factor_python.html

> Регрессия в Python

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

# способ первый
# Y = одномерный массив с ЗП, X - массив со всеми нужными нам НП

X = sm.add_constant(X)  # добавить константу, чтобы был свободный член
model = sm.OLS(Y, X)  # говорим модели, что у нас ЗП, а что НП
results = model.fit()  # строим регрессионную прямую
print(results.summary())  # смотрим результат

# способ второй, потенциально более удобный

results = smf.ols('Y ~ X1 + X2 + ... + Xn', data).fit()
print(results.summary())
```

Результаты могут выглядеть так (так же, как и с одномерной регрессией, но строчек больше):

```
In [7]: print(res.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.416
Model:                            OLS   Adj. R-squared:                  0.353
Method:                 Least Squares   F-statistic:                     6.646
Date:                Fri, 21 Feb 2020   Prob (F-statistic):            0.00157
Time:                        13:59:19   Log-Likelihood:                -12.978
No. Observations:                  32   AIC:                             33.96
Df Residuals:                      28   BIC:                             39.82
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.4639      0.162      2.864      0.008       0.132       0.796
x2             0.0105      0.019      0.539      0.594      -0.029       0.050
x3             0.3786      0.139      2.720      0.011       0.093       0.664
const         -1.4980      0.524     -2.859      0.008      -2.571      -0.425
==============================================================================
Omnibus:                        0.176   Durbin-Watson:                   2.346
Prob(Omnibus):                  0.916   Jarque-Bera (JB):                0.167
Skew:                           0.141   Prob(JB):                        0.920
Kurtosis:                       2.786   Cond. No.                         176.
==============================================================================
```

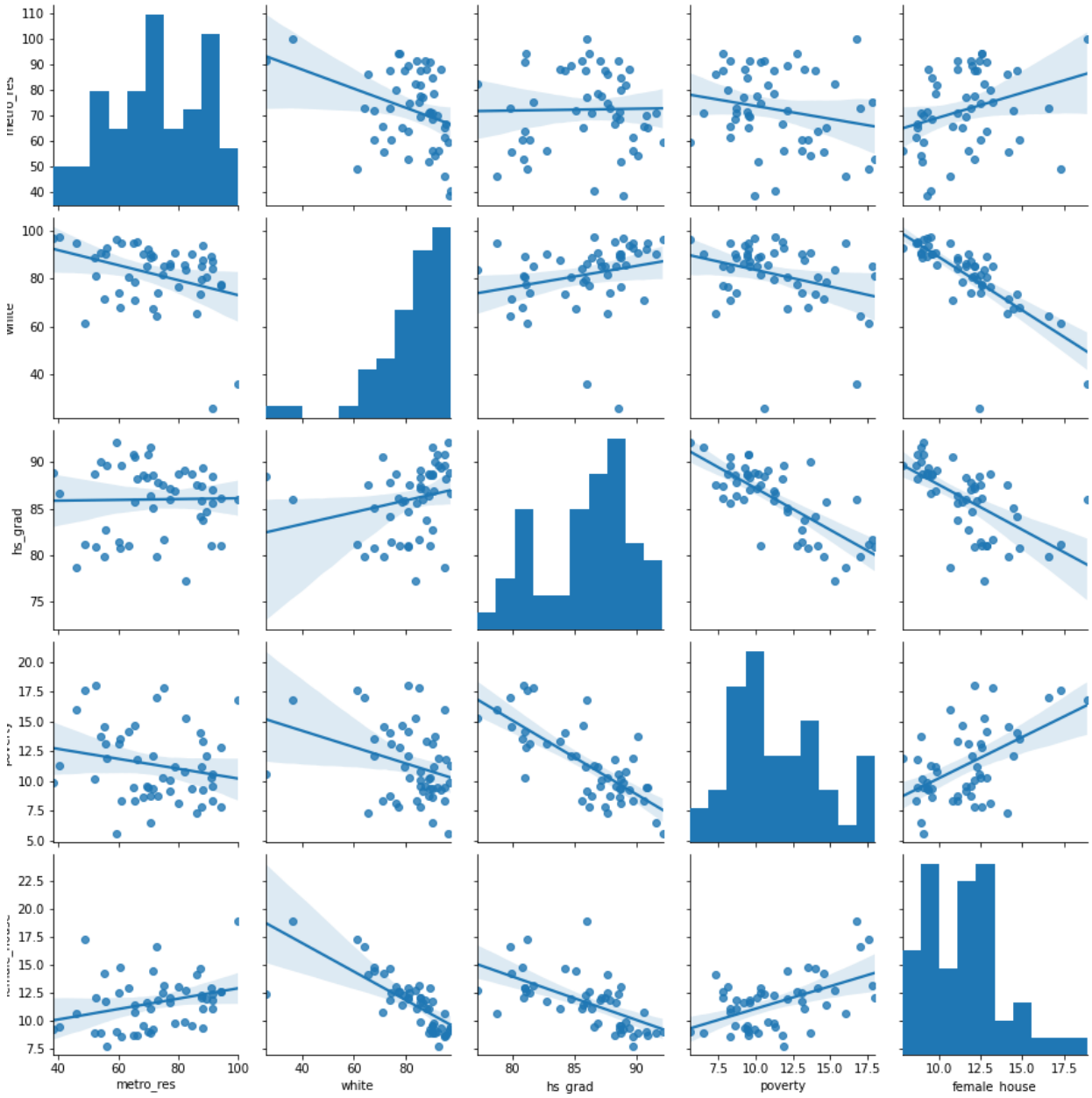
Важный момент в интерпретации результатов: каждый из коэффициентов угла наклона показывает взаимосвязь между НП и ЗП, если бы уровень остальных НП был бы нулевой. Это же иногда называют статистическим контролем: включая в модель несколько НП, мы можем раздельно оценить эффект каждой из них.

> Как нарисовать

Визуализация результатов множественной регрессии обычно крайне затруднительна ввиду большого потенциального числа переменных (максимум, что мы можем нарисовать - это три измерения). Но корреляции между всеми переменными датасета (вместе с их распределениями) можно нарисовать, например, вот так:

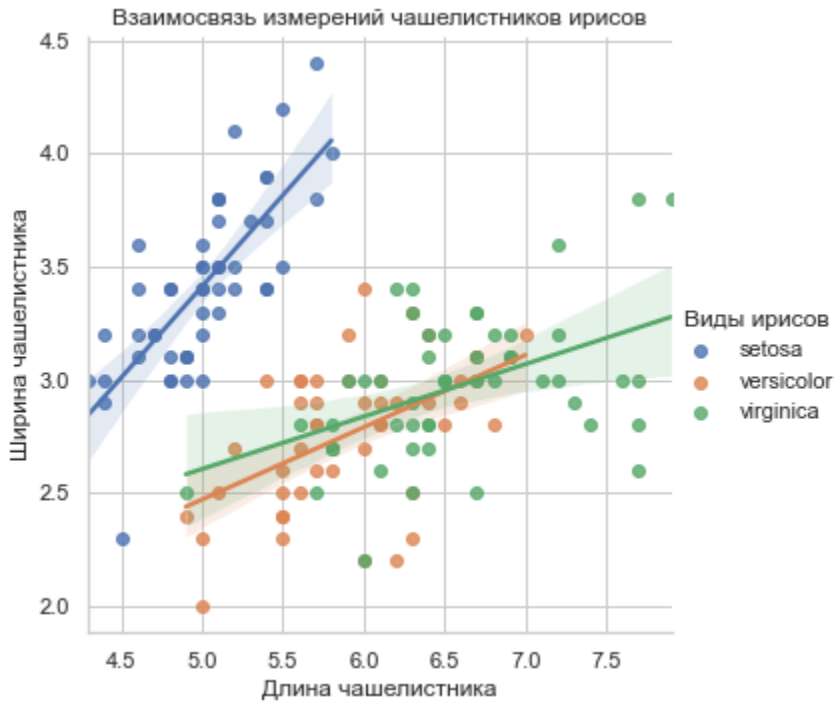
```
import seaborn as sns

sns.pairplot(data, kind = 'reg')
```



Впрочем, если какие-то из НП имеют номинативный характер, то их можно изобразить и на двумерном графике. Например, раскрасив данные по группам с помощью аргумента hue, встречающегося в некоторых функциях Seaborn:

```
g = sns.lmplot(x = 'sepal_length', y = 'sepal_width', data = iris, hue = 'species')
g._legend.set_title('Виды ирисов')
plt.title('Взаимосвязь измерений чашелистников ирисов')
plt.xlabel('Длина чашелистника')
plt.ylabel('Ширина чашелистника')
```



> Выбор наилучшей модели

Важно знать, что обычный R^2 в случае множественной регрессии менее адекватен, т.к. он растёт вне зависимости от качества НП в этой модели. Чтобы скомпенсировать этот факт, применяют исправленный R^2 (adjusted R^2) – он более остро реагирует на “бесполезные” НП, в результате чего при их добавлении R^2 снижается. Это свойство лежит в основе выбора наилучшей модели.

Типичный процесс отбора наилучшей модели – процесс **"сверху вниз"**:

1. Строим модель со всеми НП, считаем R^2

2. Убираем одну из НП, оставляем остальные, снова строим модель и оставляем R^2

3. Повторяем так для каждой, оставляем ту модель, у которой R^2 максимальный

4. Если в получившейся лучшей модели количество предикторов (НП) меньше, чем в изначальной, снова повторяем процесс уже для этой модели
5. Повторяем, пока у нас не останется модель с минимальным числом НП и максимальным R^2

Существует также вариант отбора **"снизу вверх"**:

1. Строим модель с одной НП, считаем R^2

2. Повторяем так для каждой НП, оставляем модель с максимальным R^2

3. Добавляем вторую НП из числа оставшихся
4. Повторяем для каждой оставшейся НП, оставляем модель с максимальным R^2

5. Если в получившейся лучшей модели количество НП больше, чем в изначальной, снова повторяем процесс уже для этой модели
6. Повторяем, пока у нас не останется модель с минимальным числом НП и максимальным R^2

Дополнительно:

Также существуют автоматизированные варианты вышеупомянутых подходов и их комбинации – пошаговая регрессия (stepwise regression). Однако в реальной практике этим пользоваться не рекомендуется, так как результаты могут быть крайне нестабильны и величина ошибки I рода в данном случае абсолютно непредсказуема. Подробнее о критике пошаговой регрессии – здесь.

Отбирать модели можно не только по R^2 – на практике чаще всего используют информационные критерии. Ознакомиться с ними можно, например, тут.

> Логистическая регрессия

Это расширение классической линейной регрессии, заточенное под анализ связи независимой переменной и бинарной зависимой (переменной с двумя градациями). В некотором смысле на логистическую регрессию можно взглянуть как на t-критерий наоборот (в t-критерии мы проверяем, как две группы различаются по одной количественной переменной, а в логистической регрессии мы проверяем, как одна или несколько количественных переменных влияют на возникновение одной или другой группы). Впрочем, математическая основа у этих методов принципиально разная, поэтому не стоит считать эти методы эквивалентными. Ну и не забываем про причинно-следственные связи.

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

# способ первый
# Y = одномерный массив с ЗП, X - массив со всеми нужными нам НП

X = sm.add_constant(X)  # добавить константу, чтобы был свободный член
model = sm.Logit(Y, X)  # говорим модели, что у нас ЗП, а что НП
results = model.fit()  # строим регрессионную прямую
print(results.summary())  # смотрим результат

# способ второй, потенциально более удобный
results = smf.logit('Y ~ X1 + X2 + ... + Xn', data).fit()
print(results.summary())
```

Результаты будут похожи на классическую линейную регрессию, и интерпретация коэффициентов (отчасти) сходная:

```
print(affair_mod.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                 affair   No. Observations:                 6366
Model:                          Logit   Df Residuals:                     6357
Method:                           MLE   Df Model:                            8
Date:                Fri, 21 Feb 2020   Pseudo R-squ.:                  0.1327
Time:                        13:53:46   Log-Likelihood:                -3471.5
converged:                       True   LL-Null:                       -4002.5
Covariance Type:            nonrobust   LLR p-value:                5.807e-224
==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept          3.7257      0.299     12.470      0.000       3.140       4.311
occupation         0.1602      0.034      4.717      0.000       0.094       0.227
educ              -0.0392      0.015     -2.533      0.011      -0.070      -0.009
occupation_husb    0.0124      0.023      0.541      0.589      -0.033       0.057
rate_marriage     -0.7161      0.031    -22.784      0.000      -0.778      -0.655
age               -0.0605      0.010     -5.885      0.000      -0.081      -0.040
yrs_married        0.1100      0.011     10.054      0.000       0.089       0.131
children          -0.0042      0.032     -0.134      0.893      -0.066       0.058
religious         -0.3752      0.035    -10.792      0.000      -0.443      -0.307
==============================================================================
```

Обратите внимание, что здесь R^2 называется ***pseudo*** R^2. Это аналог R^2 для логистических моделей, обладающий сходной практической интерпретацией (чем больше – тем лучше).

Обычная логистическая регрессия может работать только с бинарными зависимыми переменными. Однако на случай большего количества уровней ЗП можно применить **мультиномиальную логистическую регрессию**:

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

# способ первый
# Y = одномерный массив с ЗП, X - массив со всеми нужными нам НП

X = sm.add_constant(X)  # добавить константу, чтобы был свободный член
model = sm.MNLogit(Y, X)  # говорим модели, что у нас ЗП, а что НП
results = model.fit()  # строим регрессионную прямую
print(results.summary())  # смотрим результат

# способ второй, потенциально более удобный
results = smf.mnlogit('Y ~ X1 + X2 + ... + Xn', data).fit()
print(results.summary())
```

> Кластеризация

Мы много смотрели на ситуации, когда мы сравниваем между собой группы или предсказываем группы на основе количественных переменных. Но часто бывает так, что у нас нет готового разделения на групп и/или нам хочется найти ещё одно разделение.

Эту функцию выполняет кластерный анализ – в его рамках по нескольких количественным переменным наблюдения объединяются в группы "по схожести". Алгоритмов кластерного анализа большое количество: k-means, разные виды иерархического кластерного анализа, DBSCAN, модельно-ориентированная кластеризация и многие другие – но общая идея у всех одна.

```
from sklearn.cluster import AgglomerativeClustering

clustering = AgglomerativeClustering().fit(X)
```

> Дополнительные материалы

- Интересно немного поглубже погрузиться в математику, лежащую за линейной регрессией, особенно в контексте машинного обучения? Можно немного почитать об этом вот тут: https://habr.com/en/company/ods/blog/323890/
- Между номинативными и метрическими количественными переменными также лежит ещё один тип данных – **порядковые переменные**. Это такие данные, для которых мы можем понять их порядок изменения (какое число больше, какое меньше), но при этом не можем говорить об одинаковости расстояния между числами. Например, в соревнованиях по бегу у нас может быть понятие первого, второго и третьего места, но мы не можем определить, насколько они отличаются друг от друга по затраченному времени (возможно, между первым и вторым участником разрыв всего в 5 секунд, а третий отстаёт от них на минуту, или первый обогнал всех на минуту, а второй и третий пришли с

небольшим разрывом). Для таких данных существует **порядковая регрессия**, с которой можно ознакомиться тут: https://pythonhosted.org/mord/

- Как можете заметить, у линейной регрессии есть очень много модификаций в зависимости от характера распределения зависимой переменной. Всё это попадает под понятие **обобщённые линейные модели** (Generalized Linear Models). Ознакомиться с сутью можно тут: https://varmara.github.io/linmodr/12_GLM_gaussian.pdf
- Помимо линейной регрессии, есть много моделей, ищущих взаимодействия между количественными переменными, а помимо логистической регрессии – много моделей для классификации. Многие из них, а также алгоритмы кластеризации, сокращения размерности и другие алгоритмы машинного обучения, можно найти в библиотеке scikit-learn: https://scikit-learn.org/stable/

СТАТИСТИКА