ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

Отчет по практической работе

студента 2 курса Мукина Юрия Дмитриевича
(фамилия, имя, отчество)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 804 «Теория вероятности и компьютерное моделирование»
Учебная группа «М8О-204Б-18»
Направление 01.03.02 Прикладная математика
(шифр) (название направления)

Вид практики учебная(вычислительная)
(учебная (вычислительная/исследовательская), производственная (преддипломная))
В Московском авиационном институте (НИУ)
(наименование предприятия/учреждения/организации)

Руководитель практики от МАИ Игнатов А.Н. _____
(ФИО) (подпись)

Мукин Ю. Д. / _____ / 3 июля 2020г.
(ФИО) (подпись) (дата)

Москва 2020

Задача.

На железной дороге иногда грузовые поезда сходят с рельсов, при этом, если движение происходит на двухпутном участке, сошедшие с путей подвижные единицы могут войти в габарит соседнего пути.

Задача состоит в построении логистической регрессии между целевым полем exit и параметрами date, length, commonlength, maxder, dcar, speed, weight, load, curve, profile, type, обозначающими:

- date — месяц, в который было произведено наблюдение.
- length — количество вагонов в составе.
- commonlength — количество вагонов с секциями локомотивов.
- maxder — количество вагонов после первого сошедшего +1.
- dcar — количество сошедших вагонов.
- speed — скорость движения состава в момент схода.
- weight — вес поезда.
- load — степень загрузки поезда.
- curve — кривизна путей в месте схода.
- profile — профиль путей в месте схода.
- type — переменная обозначающая место и причину схода.

Метод решения.

Построение логистической регрессии.

Логистическая регрессия позволяет спрогнозировать вероятность возникновения некоторого события. Как правило ответ выдается в бинарном виде 0 — событие не произошло, 1 — событие произошло.

Предсказание производится по следующему алгоритму:

- Путем минимизации функции цены ошибки: $-\sum_{i=1}^{n}(y_i * \ln(P_i) + (1-y_i) * \ln(1-P_i))$ в которой y_i – исход i-го наблюдения, P_i – вероятность „положительного” исхода события, которая рассчитывается по формуле: $P = \frac{1}{(1+e^{-(\theta_0+\theta_1 x_1+\ldots+\theta_n x_n)})}$ где $x_1 \ldots x_n$ — факторы; вычисляются веса (коэффициенты) логистической регрессии $\theta_0 \ldots \theta_n$.
- Используя подобранные θ, для каждого наблюдения вычисляется значение $t = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n$, а затем определятся значение вероятности принадлежности рассматриваемого события к одной из групп (я рассматриваю формулу определяющую вероятность принадлежности группе 1, то есть группе вагонов перекрывших движение по соседнему пути) по формуле $P = f(t) = \frac{1}{1+e^{-t}}$.
- Каким либо образом подбирается пороговое значение (один из способов — применить ROC анализ) **k** и с помощью правила $\begin{cases} f(t_i) < k \to 0 \\ f(t_i) \geq k \to 1 \end{cases}$ классифицируются объекты.

В силу специфики поставленной задачи последний шаг опускается.

Реализация алгоритма в коде:

функция цены ошибки

```
#тут вводим функцию стоимости ошибки, котороую будем минимизировать, тем самым обучая модель
def Error_cost(theta, x, y):
    #функция которая при вероятности стремящейся к 1 при целевой 0 стремится к бесконечности и наоборот
    #аналогичное поведеними при стремлении к 0 и целивой 1
    cost = -np.sum(y * np.log(Probability(theta, x)) + (1 - y) * np.log(
        1 - Probability(theta, x)))
    return cost
```

вероятность входа в габарит

```
#вычисляем вероятность вхождения в группу вагонов перекрывших соседний путь
def Probability(thet, x):
    theta = thet[1:]
    tmp = np.exp(-x.mul(theta, axis = 1).sum(axis=1)-thet[0])
    #tmp = np.exp(-x.mul(thet, axis = 1).sum(axis=1))
    return 1/(1+tmp)
```

минимизация функции цены ошибки

```
#поскольку, задача минимизации функции довольно сложна и комплексна, использую готовое решение
def Get_weight(x, y, theta):
    opt_weights = fmin_tnc(func=Error_cost, x0=theta, fprime=Grad,args=(x, y))
    return opt_weights[0]
```

Для минимизации функции была использована библиотека scipy, а именно метод fmin_tnc. Этот метод требует на вход следующие параметры:

- func — функция, которую необходимо минимизировать.
- x0 — список начальных значений.
- fprime — градиент функции.
- args — аргументы функции (переменные – x, целевые значения — y)

Более подробную информацию можно найти на сайте:
https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.optimize.fmin_tnc.html

Подготовка данных.

В первую очередь нужно избавиться от наблюдений исход которых неизвестен, затем исключаются примеры в которых движение происходит по дороге с одним путем. Далее работаем с пустыми ячейками. Наиболее верным решением было бы удалить их. Избавиться от строк с значением null, если таковых в столбце меньшинство, или от столбцов, если значение null в них преобладает. Но, в силу малого объема выборки, заполним пропуски средним значением столбца, что и было сделано.

Большая часть столбцов не требует какой либо дополнительной обработки, за исключением поля data. Риск происшествия на железной дороге зависит от времени года, так, например, зимой риск того, что поезд сойдет с рельсов будет больше чем летом, из чего следует, что месяц в который было произведено наблюдение тоже можно считать фактором пригодным для построения регрессии. Год тоже можно принять за фактор, но на практике это лишь добавляет шум, поэтому в поле data остается единственное число, обозначающее номер месяца.

Реализация в коде:

```
def Get_ready(data):
    #удалим строки с колличеством путей не 2 и с неизвестным исходом
    data = data.drop(pd.isnull(data)[(data.twoways!=1) | (pd.isnull(data).exit==True)].index)
    #из ифомрации о временим происшествия оставим только месяц
    data['date'] = pd.DatetimeIndex(data['date']).month
    #заполним пропуски средним значением столбца
    data = data.fillna(data.mean())
    #data.to_csv('res.csv', sep = ',')
    return data
```

После того как данные подготовлены можно построить модель, однако какие факторы стоит использовать, а какие исключить не очевидно, в силу неопределенности их влияния на целевое значение. Исходя из этого наиболее рациональным решением будет перебрать все возможные комбинации факторов, на основе каждой из них построить по модели, а затем выбрать одну с наилучшей оценкой (о метриках качества в следующем пункте).

Перебор комбинаций был реализован с помощью модуля combinations из библиотеки itertools. Документация к itertools: https://docs.python.org/2/library/itertools.html
Реализация в коде:

```
d = ['date', 'length','commonlength','maxder','dcar','speed','weight','load'
 ,'curve','profile','type']
count0 = 0
for j in range(1,len(d)):
    for i in combinations(d, j):
        count0 = count0 +1
best = 0
count1 = 0
bs=[]
for j in range(1,len(d)):
    for i in combinations(d, j):
        count1 = count1 + 1
        lst = list(i)
        fit = data[lst].copy()
```

Оценка качества модели.

Задачу оценки качества сильно усложняет малый размер выборки. Для оценки было использовано две метрики качества:

1) AUC
2) AIC

AUC метрика.

Проведя ROC анализ модели становится возможно вычислить ее AUC характеристику. Сама по себе она малоинформативна, однако с её помощью можно сравнить несколько моделей между собой (чем больше значение AUC, тем выше качество). Сам ROC анализ определяет следующие характеристики модели:

- Чувствительность — характеристика определяющая степень реакции модели на положительные (входящие в искомую группу) примеры. Если модель будет гиперчувствительна, то она сможет определить большую часть объектов относящихся к группе 1, но при этом отнесет к ней множество лишних элементов.
- Специфичность — характеристика определяющая степень реакции модели на отрицательные результаты. Модель обладающая высокой степенью специфичности определит большую часть случаев не входящих в множество 1, однако и многие представители группы 1 будет отнесена к 0 группе.

Для вычисления чувствительности и специфичности необходимы следующие значения:

прогноз\факт	положительно	отрицательно
положительно	TP	FP
отрицательно	FN	TN

Чувствительность определяется по формуле:

$$S_e = TPR = \frac{TP}{TP + FN} * 100\,\%$$

Специфичность определяется по формуле:

$$S_p = FPR = \frac{TN}{TN + FP} * 100\,\%$$

AUC – площадь под графиком зависимости чувствительности от специфичности, исходя из этого становится очевидно, что необходимо определить не единственную пару параметров чувствительности и специфичности, а набор значений. В своем решение я добиваюсь этого перебором пороговых значений от минимальной вероятности, до максимальной из предсказанных моделью. Таким образом, получаем наборы значений Se и Sp.
Реализация AUC в коде:
площадь под графиком

```
def AUC(data):
    AUC = 0
    X,Y = Sen_Spe(data)
    for i in range(len(X)-1):
        AUC = AUC + ((X[i]+X[i+1])/2)*(Y[i]-Y[i+1])
    return(AUC)
```

вычисление специфичности и чувствительности

```
def Sen_Spe(data):
    data['exitp']=0
    t_max = max(data.my_predict)
    t =  min(data.my_predict)
    X=[]
    Y=[]
    while t<=t_max:
        data.exitp[data.my_predict>=t] = 1
        data.exitp[data.my_predict<t] = 0
        TP = ((data.exit == 1) & (data.exitp == 1)).sum()
        TN = ((data.exit == 0) & (data.exitp == 0)).sum()
        FP = ((data.exit == 0) & (data.exitp == 1)).sum()
        FN = ((data.exit == 1) & (data.exitp == 0)).sum()
        Y.append(TP/(TP+FN))
        X.append(TN/(TN+FP))
        t = t+0.01
    return X,Y
```

AIC метрика.

AIC – Информационный критерий Акаике, характеристика модели используемая исключительно для сравнения моделей (чем меньше значение, тем лучше). В отличие от коэффициента детерминации позволяет корректно сравнивать модели с разным количеством параметров Вычисляется по формуле:

$AIC = 2k - 2\ln(L)$, где k – количество параметров, L – максимизированое значение функции правдоподобия рассчитываемое по формуле $L = \prod_{i=1}^{n} p^{y_i} * (1-p)^{1-y_i}$,где $p = \frac{1}{(1+e^{-(\theta_0+\theta_1 x_1+\ldots+\theta_n x_n)})}$.

Так же функцию правдоподобия можно представить в логарифмическом виде:

$L = \sum_{i=1}^{n} (y_i * \ln(P_i) + (1-y_i) * \ln(1-P_i))$. Такой переход делается для облегчения задачи вычисления производных, что необходимо для максимизации значения функции. Несмотря на то, что сам экстремум функции меняется, точки экстремума остается неизменными. Однако, для построения метрики не нужно брать производные, поэтому можно воспользоваться обычной функцией правдоподобия.

Реализация AIC в коде:

```
def AIC(data, k):
    L = np.prod((data.my_predict**data.exit)*(1-data.my_predict)**(1-data.exit))
    return 2*k-2*np.log(L)
```

Применение.

Сначала подберем оптимальный набор по AUC:

по итогу перебора 2046 комбинаций (полный список параметров: 'date', 'length','commonlength','maxder','dcar','speed','weight','load','curve','profile','type') была получена модель с следующими параметрами:

```
AIC:  136.90029416582982
AUC:  0.8626048817696417
 ['length', 'commonlength', 'maxder', 'dcar', 'weight', 'load', 'curve', 'profile', 'type']
```

далее в качестве множества параметров X (в коде fit) записываются колонки ['length', 'commonlength', 'maxder', 'dcar', 'weight', 'load', 'curve', 'profile', 'type'] (так же запомним эти колонки для повторного использования). По какой причине лучшим оказывается вариант не со всеми возможными параметрами сложно сказать, предполагаю, что прочие параметры вносят больше шума нежели действительно полезной информации. Вновь минимизируется функция цены ошибки, рассчитываются вероятности вхождения в группу 1 с помощью формулы $P=f(t)=\frac{1}{1+e^{-t}}$ и результат записывается в колонку my_predict.

Веса:

	0
0	-0.0901026
1	-0.0250768
2	-0.0235158
3	-0.00291231
4	0.324545
5	0.000263888
6	0.00827094
7	-0.000123821
8	-0.00110003
9	0.141536

Вероятности:

```
[0.15619224 0.15208194 0.16598313 0.99183098 0.15772845 0.1946386
 0.80335058 0.17987499 0.09317748 0.12530092 0.16115487 0.22509805
 0.16393079 0.18289111 0.74926726 0.16356292 0.05504524 0.5236743
 0.32745803 0.16438061 0.15725782 0.12447783 0.70063961 0.05855288
 0.0700616  0.16542639 0.12531548 0.15023611 0.07385588 0.09883177
 0.21841699 0.04407245 0.03504607 0.06862691 0.17371187 0.01996706
 0.14121398 0.1415799  0.23521331 0.08559869 0.14218504 0.09436443
 0.37413119 0.52361203 0.01980579 0.85052232 0.97812192 0.99832164
 0.14604227 0.02276491 0.17418703 0.38621412 0.03568484 0.18575009
 0.41925335 0.04969764 0.01605318 0.08786318 0.08312899 0.20463479
 0.06604985 0.45366252 0.03017769 0.06253913 0.17306094 0.2090986
 0.17234091 0.17465116 0.19453963 0.07809062 0.02630935 0.21327111
 0.27446043 0.0947165  0.16889832 0.08726204 0.1445728  0.45168844
 0.08338211 0.05750514 0.1820958  0.16916723 0.10897858 0.07049629
 0.15536678 0.07254364 0.23324635 0.05697754 0.11213114 0.12864805
 0.15738257 0.21449705 0.99549023 0.14872649 0.04825577 0.15725465
 0.00990217 0.13044806 0.06382777 0.13260342 0.02793841 0.06040472
 0.01029197 0.02981472 0.32340019 0.07693599 0.09777484 0.10457624
 0.17215054 0.1752256  0.14453955 0.07374413 0.8501978  0.98324525
 0.45939077 0.84006753 0.09839951 0.99980796 0.99963957 0.7726316
 0.95811414 0.96824325 0.0267445  0.79098647 0.99814754 0.04546702
 0.41770001 0.99111974 0.15918531 0.07785748 0.73992363 0.98185024
 0.46476924 0.96039606 0.21172391 0.17955764 0.09473888 0.09529285
 0.15044136 0.17907532 0.62353934 0.28353805 0.23471616 0.0709772
 0.98302193 0.5121827  0.66763139 0.15445165 0.36507934 0.47964491
 0.42657873 0.04128174 0.27839287 0.84386013 0.40106389 0.46291359
 0.23532337 0.19191378 0.99999986 0.21250064]
```

Теперь попробуем перебрать все те же комбинации оценивая их при помощи AIC. По итогам подбора параметров получаем параметры:

```
AUC:  0.7718344774980932
AIC:  125.08707700390333
 ['length', 'dcar', 'weight']
```

Теперь в fit записаны колонки ['length', 'dcar', 'weight']. Не удивительно, что количество параметров уменьшилось, поскольку AIC, в отличии от AUC, штрафует за добавление параметров. Далее последовательность действий точно такая же. После оптимизации получаем следующие веса:

	0
0	-0.0068906
1	-0.0546035
2	0.330503
3	0.000302806

Вероятности в такой конфигурации модели равняются:

```
[0.15393868 0.15231691 0.16600698 0.99366644 0.16770444 0.2030536
 0.82837893 0.17639108 0.10038128 0.12998876 0.17795045 0.2462682
 0.19158189 0.20861161 0.7683309  0.1572987  0.04846308 0.56943999
 0.35571881 0.17376821 0.17557106 0.14220933 0.70733758 0.05027157
 0.06460552 0.18507216 0.1261269  0.16906444 0.06335387 0.09267917
 0.23689302 0.03838986 0.03571434 0.05712457 0.18425152 0.01515544
 0.14192879 0.16212102 0.23235772 0.07824594 0.14869144 0.08852331
 0.38872722 0.53933523 0.01975404 0.87227509 0.9815214  0.99874149
 0.16199664 0.02033345 0.17369433 0.41776216 0.02912856 0.19680929
 0.43969685 0.04711658 0.01432452 0.08733827 0.07766821 0.19520925
 0.0604383  0.48076502 0.02781932 0.05527367 0.18754987 0.20521695
 0.2150192  0.19156201 0.18573162 0.07824117 0.02576776 0.22548482
 0.28923321 0.0858866  0.19133227 0.07903813 0.16672298 0.43385125
 0.07221726 0.04959177 0.17680168 0.17541005 0.10425515 0.06232497
 0.16078875 0.06179178 0.24141421 0.05038229 0.1131926  0.12059203
 0.17865919 0.21260241 0.99642603 0.15083927 0.04581957 0.17433196
 0.00794575 0.12289067 0.05733245 0.12860554 0.02511685 0.06022333
 0.00813246 0.02527302 0.32920977 0.06586758 0.0937477  0.10060385
 0.18580398 0.2047444  0.1741609  0.06842019 0.84224345 0.98500573
 0.4226151  0.83168307 0.08796998 0.99983856 0.9997085  0.77669275
 0.95941634 0.96875818 0.01829315 0.7774925  0.99843168 0.03285222
 0.41192717 0.99133295 0.13568925 0.0676916  0.73507465 0.98357743
 0.45136409 0.96112185 0.18835171 0.15447876 0.06749956 0.06498663
 0.10878361 0.14491547 0.55944065 0.24240458 0.20614326 0.04631625
 0.98098038 0.44603302 0.63474875 0.14041076 0.32110873 0.42322417
 0.40039186 0.0246041  0.24391514 0.83228275 0.34047508 0.42914031
 0.19151975 0.15412411 0.99999987 0.16475516]
```