Yury Kashnitskiy Follow
Data Scientist at Mail.Ru Group
Apr 2 · 20 min read

Open Machine Learning Course. Topic 8. Vowpal Wabbit: Fast Learning with Gigabytes of Data

This week, we'll cover two reasons for Vowpal Wabbit's exceptional training speed, namely, online learning and hashing trick, in both theory and practice. We will try it out with news, movie reviews, and StackOverflow questions.

Outline

1. Stochastic gradient descent and online learning

- 1.1. SGD
- 1.2. Online approach to learning

2. Categorical data processing: Label Encoding, One-Hot Encoding, Hashing trick

The following content is better viewed and reproduced as a Jupyter-notebook or a Kaggle kernel.

1. Stochastic gradient descent and online learning

1.1. Stochastic gradient descent

Despite the fact that gradient descent is one of the first things learned in machine learning and optimization courses, it is one of its modifications, Stochastic Gradient Descent (SGD), that is hard to top.

Recall that the idea of gradient descent is to minimize some function by making small steps in the direction of the fastest decrease. This method was named due to the following fact from calculus: vector

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots \frac{\partial f}{\partial x_n}\right)^{\mathrm{T}}$$

of partial derivatives of the function f(x) = f(x1, …, xn) points to the direction of the fastest function growth. It means that, by moving in the opposite direction (antigradient), it is possible to decrease the function value with the fastest rate.

A snowboarder (me) in Sheregesh, Russia's most popular winter resort. Highly recommended it if you like skiing or snowboarding.

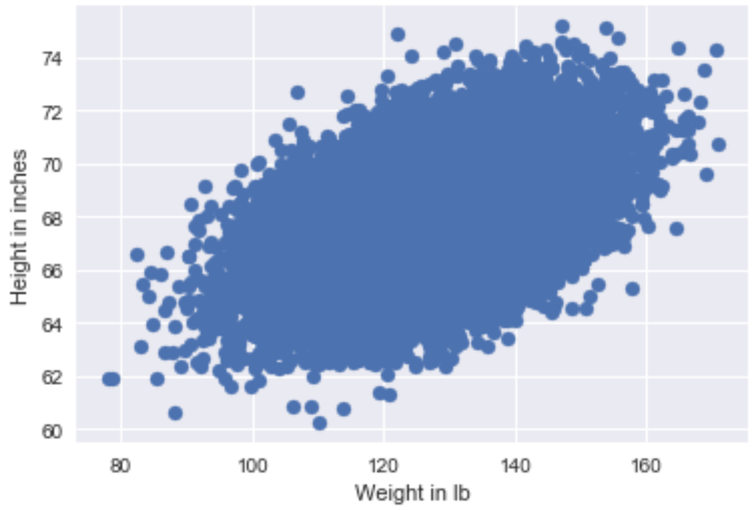
In addition to advertising beautiful landscapes, this picture depicts the idea of gradient descent. If you want to ride as fast as possible, you need to choose the path of steepest descent. Calculating antigradients can be seen as evaluating the slope at various spots.

Example

The paired regression problem can be solved with gradient descent. Let us predict one variable using another: height with weight. Assume that these variables are linearly dependent. We will use the SOCR dataset.

```
import warnings
warnings.filterwarnings('ignore')
import os
import re
import numpy as np
import pandas as pd
from tqdm import tqdm_notebook
from sklearn.datasets import fetch_20newsgroups, load_
from sklearn.preprocessing import LabelEncoder, OneHot
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, acc
```

```
PATH_TO_ALL_DATA = '../../data/'
data_demo = pd.read_csv(os.path.join(PATH_TO_ALL_DATA,
                                     'weights_heights.c
plt.scatter(data_demo['Weight'], data_demo['Height']);
plt.xlabel('Weight in lb')
```



Here we have a vector *x* of dimension *ℓ* (weight of every person i.e. training sample) and *y*, a vector containing the height of every person in the dataset.

The task is the following: find weights *w0* and *w1* such that predicting height as *yi* = *w0* + *w_1 xi* (where *yi* is *i*-th height value, *xi* is *i*-th weight value) minimizes the squared error (as well as mean squared error since $1/\ell$ doesn't make any difference):

$$SE(w_0, w_1) = \frac{1}{2}\sum_{i=1}^{\ell}(y_i - (w_0 + w_1 x_i))^2 \rightarrow min_{w_0,w_1}$$

We will use gradient descent, utilizing the partial derivatives of *SE(w0,w1)* over weights *w0* and *w1*. An iterative training procedure is then defined by simple update formulas (we change model weights in small steps, proportional to a small constant η, towards the antigradient of the function *SE(w0, w1))*:

$$w_0^{(t+1)} = w_0^{(t)} - \eta \frac{\partial SE}{\partial w_0}|_t$$
$$w_1^{(t+1)} = w_1^{(t)} - \eta \frac{\partial SE}{\partial w_1}|_t$$

Computing the partial derivatives, we get the following:

$$w_0^{(t+1)} = w_0^{(t)} + \eta \sum_{i=1}^{\ell}(y_i - w_0^{(t)} - w_1^{(t)} x_i)$$
$$w_1^{(t+1)} = w_1^{(t)} + \eta \sum_{i=1}^{\ell}(y_i - w_0^{(t)} - w_1^{(t)} x_i)x_i$$

This math works quite well as long as the amount of data is not large (we will not discuss issues with local minima, saddle points, choosing the learning rate, moments and other stuff – these topics are covered very thoroughly in the Numeric Computation chapter in "Deep Learning"). There is an issue with batch gradient descent—the gradient evaluation requires the summation of a number of values for every object from the training set. In other words, the algorithm requires a lot of iterations, and every iteration recomputes weights with formula which contains a sum over the whole training set. What happens when we have billions of training samples?

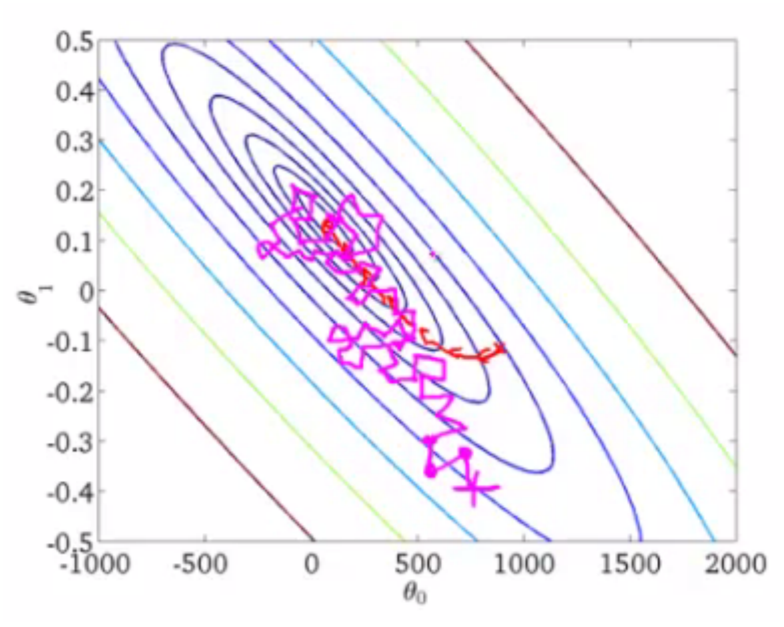
OVERFITTING... GRADIENTS EXPLODE...
I'M A NERD... XGBOOST SUCKS...
imgflip.com

Hence the motivation for stochastic gradient descent! Simply put, we throw away the summation sign and update the weights only over single training samples (or a small number of them). In our case, we have the following:

$$
\begin{aligned}
w_0^{(t+1)} &= w_0^{(t)} + \eta(y_i - w_0^{(t)} - w_1^{(t)} x_i) \\
w_1^{(t+1)} &= w_1^{(t)} + \eta(y_i - w_0^{(t)} - w_1^{(t)} x_i) x_i
\end{aligned}
$$

With this approach, there is no guarantee that we will move in best possible direction at every iteration. Therefore, we may need many more iterations, but we get much faster weight updates.

Andrew Ng has a good illustration of this in his machine learning course. Let’s take a look.



These are the contour plots for some function, and we want to find the global minimum of this function. The red curve shows weight changes (in this picture, *θ0* and *θ1* correspond to our *w0* and *w1*). According to the properties of a gradient, the direction of change at every point is orthogonal to contour plots. With stochastic gradient descent, weights are changing in a less predictable manner, and it even may seem that some steps are wrong by leading away from minima; however, both procedures converge to the same solution.

1.2. Online approach to learning

Stochastic gradient descent gives us practical guidance for training both classifiers and regressors with large amounts of data up to hundreds of GBs (depending on computational resources).

Considering the case of paired regression, we can store the training data set *(X,y)* in HDD without loading it into RAM (where it simply won't fit), read objects one by one, and update the weights of our model:

$$w_0^{(t+1)} = w_0^{(t)} + \eta(y_i - w_0^{(t)} - w_1^{(t)} x_i)$$
$$w_1^{(t+1)} = w_1^{(t)} + \eta(y_i - w_0^{(t)} - w_1^{(t)} x_i)x_i$$

After working through the whole training dataset, our loss function (for example, quadratic squared root error in regression or logistic loss in classification) will decrease, but it usually takes dozens of passes over the training set to make the loss small enough.

This approach to learning is called **online learning**, and this name emerged even before machine learning MOOC-s turned mainstream.

Andrew Ng knew about online learning
Long before setting this mainstream

We did not discuss many specifics about SGD here. If you want dive into theory, I highly recommend “Convex Optimization” by Stephen Boyd. Now, we will introduce the Vowpal Wabbit library, which is good for training simple models with huge data sets thanks to stochastic optimization and another trick, feature hashing.

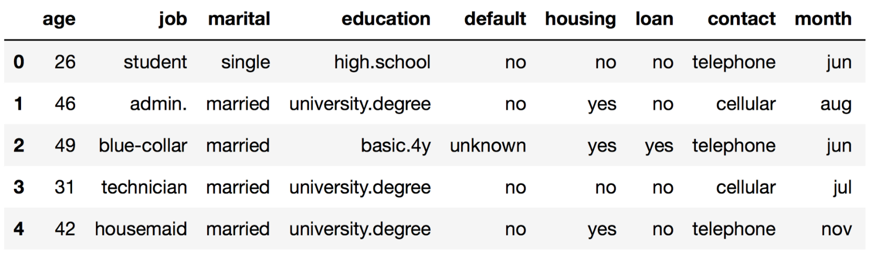
In scikit-learn, classifiers and regressors trained with SGD are named `SGDClassifier` and `SGDRegressor` in `sklearn.linear_model`. These are nice implementations of SGD, but we'll focus on VW since it is more performant than sklearn's SGD models in many aspects.

2. Categorical feature processing: Label Encoding, One-Hot Encoding, and Hashing trick

Many classification and regression algorithms operate in Euclidean or metric space, implying that data is represented with vectors of real numbers. However, in real data, we often have categorical features with discrete values such as yes/no or January/February/…/December. We will see how to process this kind of data, particularly with linear models, and how to deal with many categorial features even when they have many unique values.

Let's explore the UCI bank marketing dataset where most of features are categorial. Actually, more features are present in this dataset, explore this Jupyter notebook.

```
df = pd.read_csv(os.path.join(PATH_TO_ALL_DATA, 'bank_t
labels = pd.read_csv(os.path.join(PATH_TO_ALL_DATA,
                                  'bank_train_target.cs

```

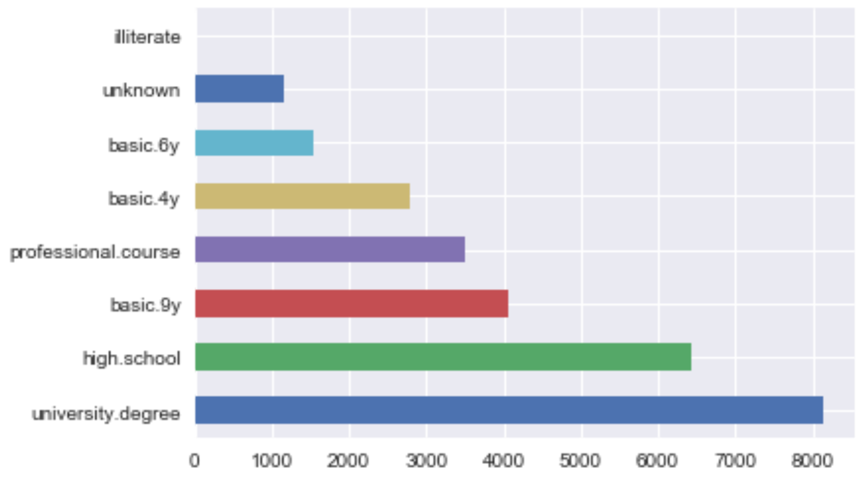


	age	job	marital	education	default	housing	loan	contact	month
0	26	student	single	high.school	no	no	no	telephone	jun
1	46	admin.	married	university.degree	no	yes	no	cellular	aug
2	49	blue-collar	married	basic.4y	unknown	yes	yes	telephone	jun
3	31	technician	married	university.degree	no	no	no	cellular	jul
4	42	housemaid	married	university.degree	no	yes	no	telephone	nov

We can see that most of features are not represented by numbers. This poses a problem because we cannot use most machine learning methods (at least those implemented in `scikit-learn`) out-of-the-box.

Let's dive into the "education" feature.

```
df['education'].value_counts().plot.barh();
```



The most straightforward solution is to map each value of this feature into a unique number. For example, we can map `university.degree` to 0, `basic.9y` to 1, and so on. You can use `sklearn.preprocessing.LabelEncoder` to perform this mapping.

```
label_encoder = LabelEncoder()
```

The `fit` method of this class finds all unique values and builds the actual mapping between categories and numbers, and the `transform` method converts the categories into numbers. After `fit` is executed, `label_encoder` will have the `classes_` attribute with all unique values of the feature. Let us count them to make sure the transformation was correct.

```
mapped_education = pd.Series(label_encoder.fit_transfor
    df['education']))
mapped_education.value_counts().plot.barh()
print(dict(enumerate(label_encoder.classes_)))
```

```
# Output:

{0: 'basic.4y', 1: 'basic.6y', 2: 'basic.9y', 3:
'high.school', 4: 'illiterate', 5: 'professional.course', 6:
'university.degree', 7: 'unknown'}
```

```
df['education'] = mapped_education df.head()
```

	age	job	marital	education	default	housing	loan	contact	month
0	26	student	single	3	no	no	no	telephone	jun
1	46	admin.	married	6	no	yes	no	cellular	aug
2	49	blue-collar	married	0	unknown	yes	yes	telephone	jun
3	31	technician	married	6	no	no	no	cellular	jul
4	42	housemaid	married	6	no	yes	no	telephone	nov

Let's apply the transformation to other columns of type `object` .

```
categorical_columns = df.columns[df.dtypes
                                 == 'object'].union(['e
for column in categorical_columns:
    df[column] = label_encoder.fit_transform(df[column]
```

	age	job	marital	education	default	housing	loan	contact	month
0	26	8	2	3	0	0	0	1	4
1	46	0	1	6	0	2	0	0	1
2	49	1	1	0	1	2	2	1	4
3	31	9	1	6	0	0	0	0	3
4	42	3	1	6	0	2	0	1	7

The main issue with this approach is that we have now introduced some relative ordering where it might not exist.

For example, we implicitly introduced algebra over the values of the job feature where we can now substract the job of client #2 from the job of client #1 :

```
df.loc[1].job - df.loc[2].job
# -1.0
```

Does this operation make any sense? Not really. Let’s try to train logisitic regression with this feature transformation.

```
def logistic_regression_accuracy_on(dataframe, labels)
    features = dataframe.as_matrix()
    train_features, test_features, train_labels, test_
        train_test_split(features, labels)

    logit = LogisticRegression()
    logit.fit(train_features, train_labels)
    return classification_report(test_labels,
                                 logit.predict(test_fe
```

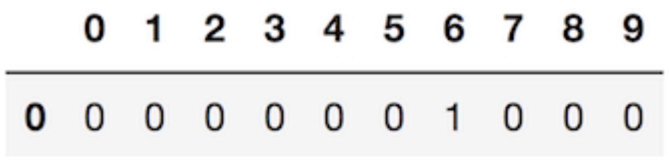
```
             precision    recall  f1-score   support

          0       0.88      1.00      0.94      6104
          1       0.50      0.00      0.00       795

avg / total       0.84      0.88      0.83      6899
```

We can see that logistic regression never predicts class 1. In order to use linear models with categorial features, we will use a different approach: One-Hot Encoding.

2.2. One-Hot Encoding

Suppose that some feature can have one of 10 unique values. One-hot encoding creates 10 new features corresponding to these unique values, all of them *except one* are zeros.

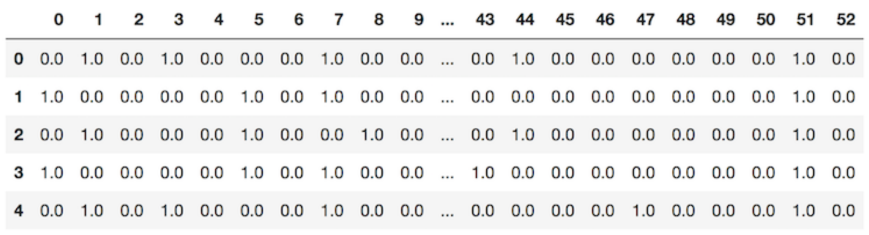
```
one_hot_example = pd.DataFrame([{i: 0 for i in range(10
one_hot_example.loc[0, 6] = 1
one_hot_example
```



	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	1	0	0	0

This idea is implemented in the `OneHotEncoder` class from `sklearn.preprocessing`. By default `OneHotEncoder` transforms data into a sparse matrix to save memory space because most of the values are zeroes and because we do not want to take up more RAM. However, in this particular example, we do not encounter such problems, so we are going to use a "dense" matrix representation.

```
onehot_encoder = OneHotEncoder(sparse=False)
encoded_categorical_columns = \
pd.DataFrame(onehot_encoder.fit_transform(
    df[categorical_columns]))
```



	0	1	2	3	4	5	6	7	8	9	...	43	44	45	46	47	48	49	50	51	52
0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
2	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
4	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0

We have 53 columns that correspond to the number of unique values of categorical features in our data set. When transformed with One-Hot Encoding, this data can be used with linear models:

```
print(logistic_regression_accuracy_on(encoded_categorical_co
lumns,  labels))
```

```
             precision    recall  f1-score   support

          0       0.90      0.99      0.94      6102
          1       0.67      0.18      0.29       797

avg / total       0.88      0.90      0.87      6899
```

2.3. Hashing trick

Real data can be volatile, meaning we cannot guarantee that new values of categorial features will not occur. This issue hampers using a trained model on new data. Besides that, `LabelEncoder` requires preliminary analysis of the whole dataset and storage of constructed mappings in memory, which makes it difficult to work with large datasets.

There is a simple approach to vectorization of categorial data based on hashing and is known as, not-so-surprisingly, the hashing trick.

Hash functions can help us find unique codes for different feature values, for example:

```python
for s in ('university.degree', 'high.school', 'illitera
    print(s, '->', hash(s))
```

```
# university.degree -> -6241459093488141593
# high.school -> 7728198035707179500
# illiterate -> -7360093633803373451
```

We will not use negative values or values of high magnitude, so we restrict the range of values for the hash function:

```python
hash_space = 25
for s in ('university.degree', 'high.school', 'illitera
    print(s, '->', hash(s) % hash_space)
```

```
# university.degree -> 7
# high.school -> 0
# illiterate -> 24
```

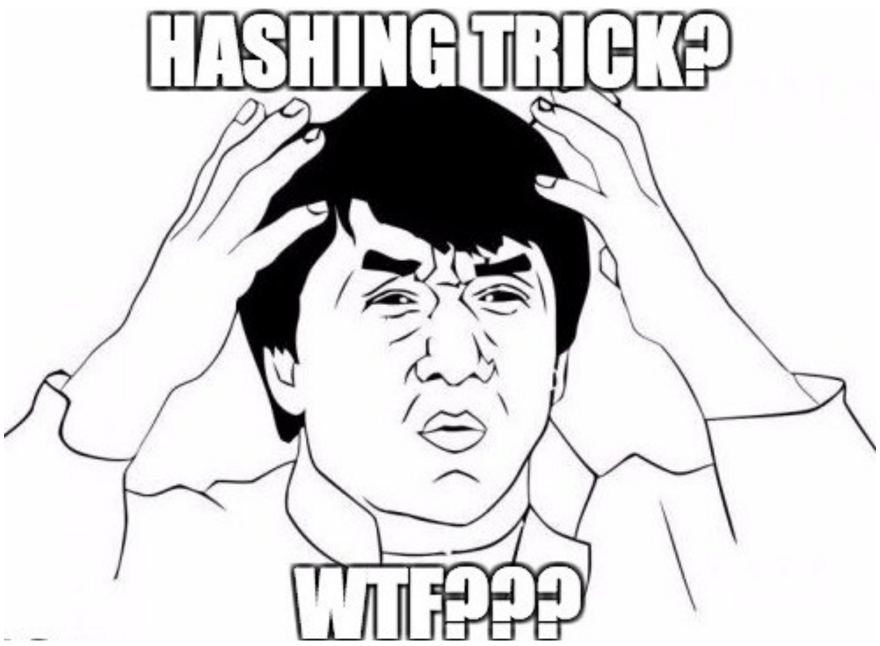
Imagine that our data set contains a single (i.e. not married) student, who received a call on Monday. His feature vectors will be created similarly as in the case of One-Hot Encoding but in the space with fixed range for all features:

```
hashing_example = pd.DataFrame([{i: 0.0 for i in range(
for s in ('job=student', 'marital=single', 'day_of_week
    print(s, '->', hash(s) % hash_space)
    hashing_example.loc[0, hash(s) % hash_space] = 1
```

```
# job=student -> 20
# marital=single -> 23
# day_of_week=mon -> 9
```

We want to point out that we hash not only feature values but also pairs of **feature name + feature value**. It is important to do this so that we can distinguish the same values of different features.

Is it possible to have a collision when using hash codes? Sure, it is possible, but it is a rare case with large enough hashing spaces. Even if collision occurs, regression or classification metrics will not suffer much. In this case, hash collisions work as a form of regularization.



You may be saying "WTF?"; hashing seems counterintuitive. This is true, but these heuristics sometimes are, in fact, the only plausible approach to work with categorial data. Moreover, this technique has proven to just work. As you work more with data, you may see this for yourself.

3. Vowpal Wabbit

Vowpal Wabbit (VW) is one of the most widespread machine learning libraries used in industry. It is prominent for its training speed and support of many training modes, especially for online learning with big and high-dimentional data. This is one of the major merits of the library. Also, with the hashing trick implemented, Vowpal Wabbit is a perfect choice for working with text data.

Shell is the main interface for VW. Try this command:

```
!vw — help
```

Vowpal Wabbit reads data from files or from standard input stream (stdin) with the following format:

```
[Label] [Importance] [Tag]|Namespace Features |Namespace
Features ... |Namespace Features
```

```
Namespace=String[:Value]
```

```
Features=(String[:Value] )*
```

here [] denotes non-mandatory elements, and (…)* means multiple inputs allowed.

- **Label** is a number. In the case of classification, it is usually 1 and -1; for regression, it is a real float value
- **Importance** is a number. It denotes the sample weight during training. Setting this helps when working with imbalanced data.
- **Tag** is a string without spaces. It is the "name" of the sample that VW saves upon prediction. In order to separate Tag from Importance, it is better to start Tag with the ‘ character.
- **Namespace** is for creating different feature spaces.

- **Features** are object features inside a given **Namespace**. Features have weight 1.0 by default, but it can be changed, for example feature:0.1.

The following string matches the VW format:

```
1 1.0 |Subject WHAT car is this |Organization University of
Maryland:0.5 College Park
```

Let's check the format by running VW with this training sample:

```
! echo '1 1.0 |Subject WHAT car is this |Organization
University of Maryland:0.5 College Park' | vw

# Output:

Num weight bits = 18
learning rate = 0.5
initial_t = 0
power_t = 0.5
using no cache
Reading datafile =
num sources = 1
average  since         example        example  current
current  current
loss     last          counter         weight    label
predict features
1.000000 1.000000            1            1.0   1.0000
0.0000       10

finished run
number of examples per pass = 1
passes used = 1
weighted example sum = 1.000000
weighted label sum = 1.000000
average loss = 1.000000
best constant = 1.000000
best constant's loss = 0.000000
total feature number = 10
```

VW is a wonderful tool for working with text data. We'll illustrate it with the 20newsgroups dataset, which contains letters from 20 different newsletters.

3.1. News. Binary classification.

```
# load data with sklearn's function
newsgroups = fetch_20newsgroups(PATH_TO_ALL_DATA)
newsgroups['target_names']
```

The 20 topics are `['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']`

Lets look at the first document in this collection:

```
text = newsgroups['data'][0]
target = newsgroups['target_names'][newsgroups['target'

print('-----')
print(target)
print('-----')
```

```
# Output:

-----
rec.autos
-----
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

 I was wondering if anyone out there could enlighten me on
this car I saw
the other day. It was a 2-door sports car, looked to be from
the late 60s/
early 70s. It was called a Bricklin. The doors were really
small. In addition,
the front bumper was separate from the rest of the body.
```

```
This is
all I know. If anyone can tellme a model name, engine specs,
years
of production, where this car is made, history, or whatever
info you
have on this funky looking car, please e-mail.

Thanks,
- IL
   ---- brought to you by your neighborhood Lerxst ----
----
```

Now we convert the data into something Vowpal Wabbit can understand. We will throw away words shorter than 3 symbols. Here, we will skip some important NLP stages such as stemming and lemmatization; however, we will later see that VW solves the problem even without these steps.

```
def to_vw_format(document, label=None):
    return str(label or '') + ' |text ' + ' '.join(re.f
                                                     document
```

```
# Output:

'1 |text from lerxst wam umd edu where thing subject what
car this nntp posting host rac3 wam umd edu organization
university maryland college park lines was wondering anyone
out there could enlighten this car saw the other day was
door sports car looked from the late 60s early 70s was
called bricklin the doors were really small addition the
front bumper was separate from the rest the body this all
know anyone can tellme model name engine specs years
production where this car made history whatever info you
have this funky looking car please mail thanks brought you
your neighborhood lerxst\n'
```

We split the dataset into train and test and write these into separate files. We will consider a document as positive if it corresponds to **rec.autos**. Thus, we are constructing a model which distinguishes articles about cars from other topics:

```
all_documents = newsgroups['data']
all_targets = [1 if newsgroups['target_names'][target]
                  else -1 for target in newsgroups['targe

train_documents, test_documents, train_labels, test_la
    train_test_split(all_documents, all_targets, rando

with open(os.path.join(PATH_TO_ALL_DATA, '20news_train
    for text, target in zip(train_documents, train_lab
```

Now, we pass the created training file to Vowpal Wabbit. We solve the classification problem with a hinge loss function (linear SVM). The trained model will be saved in the `20news_model.vw` file:

```
!vw -d $PATH_TO_ALL_DATA/20news_train.vw \ --loss_function
hinge -f $PATH_TO_ALL_DATA/20news_model.vw
```

```
# Output:
```

```
final_regressor = ../../data//20news_model.vw
Num weight bits = 18
learning rate = 0.5
initial_t = 0
power_t = 0.5
using no cache
Reading datafile = ../../data//20news_train.vw
num sources = 1
average  since         example        example  current
current  current
loss     last          counter         weight    label
predict features
1.000000 1.000000            1            1.0  -1.0000
0.0000      157
0.911276 0.822551            2            2.0  -1.0000
-0.1774      159
0.605793 0.300311            4            4.0  -1.0000
-0.3994       92
0.419594 0.233394            8            8.0  -1.0000
-0.8167      129
0.313998 0.208402           16           16.0  -1.0000
-0.6509      108
0.196014 0.078029           32           32.0  -1.0000
-1.0000      115
0.183158 0.170302           64           64.0  -1.0000
-0.7072      114
0.261046 0.338935          128          128.0   1.0000
-0.7900      110
0.262910 0.264774          256          256.0  -1.0000
-0.6425       44
0.216663 0.170415          512          512.0  -1.0000
```

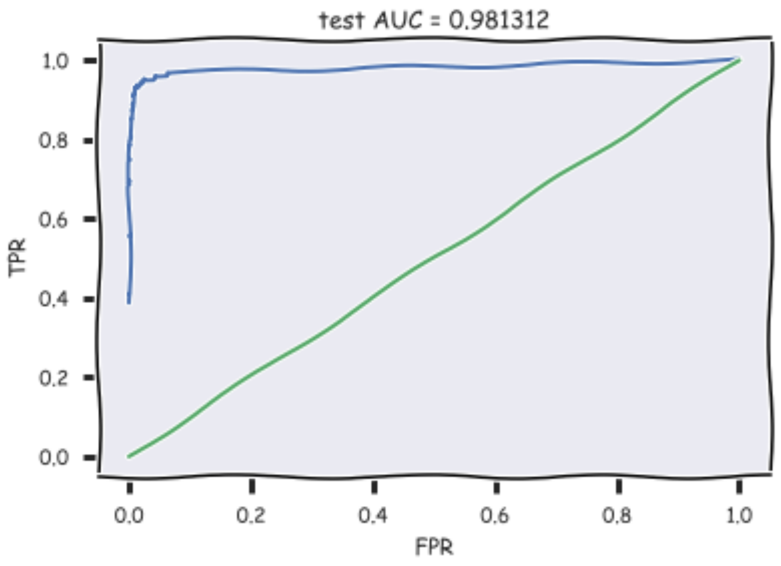
```
-1.0000      160
0.176710 0.136757         1024         1024.0  -1.0000
-1.0000      194
0.134541 0.092371         2048         2048.0  -1.0000
-1.0000      438
0.104403 0.074266         4096         4096.0  -1.0000
-1.0000      644
0.081329 0.058255         8192         8192.0  -1.0000
-1.0000      174

finished run
number of examples per pass = 8485
passes used = 1
weighted example sum = 8485.000000
weighted label sum = -7555.000000
average loss = 0.079837
best constant = -1.000000
best constant's loss = 0.109605
total feature number = 2048932
```

VW prints a lot of interesting info while training (one can suppress it with the `--quiet` parameter). You can see documentation of the diagnostic output on GitHub. Note how average loss drops while training. For loss computation, VW uses samples it has never seen before, so this measure is usually accurate. Now, we apply our trained model to the test set, saving predictions into a file with the `-p` flag:

```
!vw -i $PATH_TO_ALL_DATA/20news_model.vw -t -d
$PATH_TO_ALL_DATA/20news_test.vw \ -p
$PATH_TO_ALL_DATA/20news_test_predictions.txt
```

Now we load our predictions, compute AUC, and plot the ROC curve:

```
with open(os.path.join(PATH_TO_ALL_DATA,
                       '20news_test_predictions.txt'))
    test_prediction = [float(label)
                       for label in pred_file.readline

auc = roc_auc_score(test_labels, test_prediction)
roc_curve = roc_curve(test_labels, test_prediction)

with plt.xkcd():
    plt.plot(roc_curve[0], roc_curve[1]);
```



The AUC value we get shows that we have achieved high classification quality.

3.2. News. Multiclass classification

We will use the same news dataset, but, this time, we will solve a multiclass classification problem. `Vowpal Wabbit` is a little picky – it wants labels starting from 1 till K, where K – is the number of classes in the classification task (20 in our case). So we will use LabelEncoder and add 1 afterwards (recall that `LabelEncoder` maps labels into range from 0 to K-1).

```
all_documents = newsgroups['data']
topic_encoder = LabelEncoder()
all_targets_mult = topic_encoder.fit_transform(newsgrou
```

The data is the same, but we have changed the labels, `train_labels_mult` and `test_labels_mult`, into label vectors from 1 to 20.

```
train_documents, test_documents, train_labels_mult, te
    train_test_split(all_documents, all_targets_mult,

with open(os.path.join(PATH_TO_ALL_DATA,
                       '20news_train_mult.vw'), 'w') a
    for text, target in zip(train_documents, train_lab
        vw_train_data.write(to_vw_format(text, target)
with open(os.path.join(PATH_TO_ALL_DATA,
```

We train Vowpal Wabbit in multiclass classification mode, passing the `oaa` parameter("one against all") with the number of classes. Also, let's see what parameters our model quality is dependent on (more info can be found in the official Vowpal Wabbit tutorial):

- learning rate (-l, 0.5 default)—rate of weight change on every step
- learning rate decay (—power_t, 0.5 default)—it is proven in practice, that, if the learning rate drops with the number of steps in stochastic gradient descent, we approach the minimum loss better
- loss function (—loss_function)—the entire training algorithm depends on it. See docs for loss functions
- Regularization (-l1)—note that VW calculates regularization for every object. That is why we usually set regularization values to about 1e-20.

Additionally, we can try automatic Vowpal Wabbit parameter tuning with Hyperopt.

```
!vw — oaa 20 $PATH_TO_ALL_DATA/20news_train_mult.vw -f
$PATH_TO_ALL_DATA/20news_model_mult.vw \
 — loss_function=hinge
```

```
!vw -i $PATH_TO_ALL_DATA/20news_model_mult.vw -t -d
$PATH_TO_ALL_DATA/20news_test_mult.vw \
-p $PATH_TO_ALL_DATA/20news_test_predictions_mult.txt
```

```
with open(os.path.join(PATH_TO_ALL_DATA,
                       '20news_test_predictions_mult.tx
    test_prediction_mult = [float(label)
                            for label in pred_file.reac

accuracy_score(test_labels_mult, test_prediction_mult)
```

Here is how often the model misclassifies atheism with other topics:

```
M = confusion_matrix(test_labels_mult, test_prediction_
for i in np.where(M[0,:] > 0)[0][1:]:
    print(newsgroups['target_names'][i], M[0,i])
```

```
# Output:

rec.autos 1
rec.sport.baseball 1
sci.med 1
soc.religion.christian 3
talk.religion.misc 5
```

3.3. IMDB movie reviews

In this part we will do binary classification of IMDB (International Movie DataBase) movie reviews. We will see how fast Vowpal Wabbit performs.

Using the `load_files` function from `sklearn.datasets` , we load the movie reviews here. If you want to reproduce the results, please download the archive, unzip it, and set the path to `imdb_reviews` (it already contains *train* and *test* subdirectories). Unpacking can take several minutes as there are 100k files. Both train and test sets hold 12.5k good and bad movie reviews. First, we split the texts and labels.

```
import pickle
# change this for your path to imdb_reviews
path_to_movies = os.path.expanduser('imdb_reviews')
reviews_train = load_files(os.path.join(path_to_movies
text_train, y_train = reviews_train.data, reviews_trai

print("Number of documents in training data: %d" % len
print(np.bincount(y_train))
# Number of documents in training data: 25000
# [12500 12500]

# Do the same for the test set.
reviews_test = load_files(os.path.join(path_to_movies,
```

Take a look at examples of reviews and their corresponding labels.

```
text_train[0]
```

```
# Output:
b"Zero Day leads you to think, even re-think why two
boys/young men would do what they did - commit mutual
suicide via slaughtering their classmates. It captures what
must be beyond a bizarre mode of being for two humans who
have decided to withdraw from common civility in order to
define their own/mutual world via coupled destruction.<br />
<br />It is not a perfect movie but given what money/time
the filmmaker and actors had - it is a remarkable product.
In terms of explaining the motives and actions of the two
young suicide/murderers it is better than 'Elephant' - in
terms of being a film that gets under our 'rationalistic'
skin it is a far, far better film than almost anything you
are likely to see. <br /><br />Flawed but honest with a
terrible honesty."
```

```
y_train[0] # good review
```

```
# Output:
1
```

```
text_train[1]
```

```
# Output:
b'Words can\'t describe how bad this movie is. I can\'t
explain it by writing only. You have too see it for yourself
to get at grip of how horrible a movie really can be. Not
that I recommend you to do that. There are so many
clich\xc3\xa9s, mistakes (and all other negative things you
can imagine) here that will just make you cry. To start with
the technical first, there are a LOT of mistakes regarding
the airplane. I won\'t list them here, but just mention the
coloring of the plane. They didn\'t even manage to show an
airliner in the colors of a fictional airline, but instead
used a 747 painted in the original Boeing livery. Very bad.
The plot is stupid and has been done many times before, only
much, much better. There are so many ridiculous moments here
that i lost count of it really early. Also, I was on the bad
guys\' side all the time in the movie, because the good guys
were so stupid. "Executive Decision" should without a doubt
be you\'re choice over this one, even the "Turbulence"-
movies are better. In fact, every other movie in the world
is better than this one.'
```

```
y_train[1] # bad review


# Output:
0
```

Now, we prepare training (`movie_reviews_train.vw`), validation (`movie_reviews_valid.vw`), and test (`movie_reviews_test.vw`) sets for Vowpal Wabbit. We will use 70% for training, 30% for the hold-out set.

```
train_share = int(0.7 * len(text_train))
train, valid = text_train[:train_share], text_train[tr
train_labels, valid_labels = y_train[:train_share], y_
print(len(train_labels), len(valid_labels))
# (17500, 7500)

with open(os.path.join(PATH_TO_ALL_DATA, 'movie_review
    for text, target in zip(train, train_labels):
        vw_train_data.write(to_vw_format(str(text), 1
with open(os.path.join(PATH_TO_ALL_DATA, 'movie_review
```

Now, launch Vowpal Wabbit with the following arguments:

- -d, path to training set (corresponding .vw file)
- — loss_function—hinge (feel free to experiment here)
- -f—path to the output file (which can also be in the .vw format)

```
!vw -d $PATH_TO_ALL_DATA/movie_reviews_train.vw --
loss_function hinge -f
$PATH_TO_ALL_DATA/movie_reviews_model.vw --quiet
```

Next, make the hold-out prediction with the following VW arguments:

- -i –path to the trained model (.vw file)
- -t -d—path to hold-out set (.vw file)
- -p—path to a txt-file where the predictions will be stored

```
!vw -i $PATH_TO_ALL_DATA/movie_reviews_model.vw -t \
-d $PATH_TO_ALL_DATA/movie_reviews_valid.vw -p
$PATH_TO_ALL_DATA/movie_valid_pred.txt --quiet
```

Read the predictions from the text file and estimate the accuracy and ROC AUC. Note that VW prints probability estimates of the +1 class. These estimates are distributed from -1 to 1, so we can convert these into binary answers, assuming that positive values belong to class 1.

```
with open(os.path.join(PATH_TO_ALL_DATA, 'movie_valid_p
    valid_prediction = [float(label)
                             for label in pred_file.rea
print("Accuracy: {}".format(round(accuracy_score(valid_
                 [int(pred_prob > 0) for pred_prob in val
```

```
Accuracy: 0.885
AUC: 0.942
```

Again, do the same for the test set.

```
!vw -i $PATH_TO_ALL_DATA/movie_reviews_model.vw -t \ -d
$PATH_TO_ALL_DATA/movie_reviews_test.vw \ -p
$PATH_TO_ALL_DATA/movie_test_pred.txt --quiet
```

```
with open(os.path.join(PATH_TO_ALL_DATA, 'movie_test_pr
    test_prediction = [float(label)
                             for label in pred_file.rea
print("Accuracy: {}".format(round(accuracy_score(y_test
                 [int(pred_prob > 0) for pred_prob in tes
```

```
Accuracy: 0.88
AUC: 0.94
```

Let’s try to achieve a higher accuracy by incorporating bigrams.

```
!vw -d $PATH_TO_ALL_DATA/movie_reviews_train.vw \ --
loss_function hinge --ngram 2 -f
$PATH_TO_ALL_DATA/movie_reviews_model2.vw --quiet

!vw -i$PATH_TO_ALL_DATA/movie_reviews_model2.vw -t -d
$PATH_TO_ALL_DATA/movie_reviews_valid.vw \ -p
$PATH_TO_ALL_DATA/movie_valid_pred2.txt --quiet

!vw -i $PATH_TO_ALL_DATA/movie_reviews_model2.vw -t -d
$PATH_TO_ALL_DATA/movie_reviews_test.vw \ -p
$PATH_TO_ALL_DATA/movie_test_pred2.txt --quiet
```

Adding bigrams really helped to improve our model!

```
# Validation
Accuracy: 0.894
AUC: 0.954

# Test
Accuracy: 0.888
AUC: 0.952
```

3.4. Classifying gigabytes of StackOverflow questions

Now, let’s see Vowpal Wabbit work on large datasets. There is a 10GB dataset of StackOverflow questions here. The original dataset is comprised of 10 million questions; each question can have several tags. The data is quite clean, so don’t call it “Big Data”, even in a pub. :)

We chose only 10 tags: ‘javascript’, ‘java’, ‘python’, ‘ruby’, ‘php’, ‘c++’, ‘c#’, ‘go’, ‘scala’ and ‘swift’. Let’s solve the 10-class classification problem: we want to predict a tag corresponding to one of these 10 popular programming languages given only the text of the question. Your task in assignment is to

```
# change the path to data
PATH_TO_STACKOVERFLOW_DATA = 'stackoverflow_10mln'

# Print the first 3 lines from a sample of the dataset.

!head -3 $PATH_TO_STACKOVERFLOW_DATA/stackoverflow_train.vw

# Output:

1 | i ve got some code in window scroll that checks if an
element is visible then triggers another function however
only the first section of code is firing both bits of code
work in and of themselves if i swap their order whichever is
on top fires correctly my code is as follows fn isonscreen
function use strict var win window viewport top win
scrolltop left win scrollleft bounds this offset viewport
```

```
right viewport left + win width viewport bottom viewport top
+ win height bounds right bounds left + this outerwidth
bounds bottom bounds top + this outerheight return viewport
right lt bounds left viewport left gt bounds right viewport
bottom lt bounds top viewport top gt bounds bottom window
scroll function use strict var load_more_results ajax
load_more_results isonscreen if load_more_results true
loadmoreresults var load_more_staff ajax load_more_staff
isonscreen if load_more_staff true loadmorestaff what am i
doing wrong can you only fire one event from window scroll i
assume not
4 | redefining some constant in ruby ex foo bar generates
the warning already initialized constant i m trying to write
a sort of reallyconstants module where this code should have
this behaviour reallyconstants define_constant foo bar gt
sets the constant reallyconstants foo to bar reallyconstants
foo gt bar reallyconstants foo foobar gt this should raise
an exception that is constant redefinition should generate
an exception is that possible 1 | in my form panel i added a
checkbox setting stateful true stateid loginpanelremeberme
then before sending form i save state calling this savestate
on the panel all other componenets save their state and whe
i reload the page they recall the previous state but
checkbox alway start in unchecked state is there any way to
force save value
```

After selecting our 10 tags, we have a 4.7G dataset that we divide into train and test.

```
!du -hs $PATH_TO_STACKOVERFLOW_DATA/stackoverflow_*.vw


# Output:
```

```
4,7G
stackoverflow_10mln.vw
1,6G
stackoverflow_test.vw
3,1G
stackoverflow_train.vw
```

We will process the training set (3.1 GiB) with Vowpal Wabbit and the following arguments:

- -oaa 10—for multiclass classification with 10 classes
- -d—path to data
- -f—path to output file of the trained model

- -b 28—we will use 28 bits for hashing, resulting in a $2^{28}$-sized feature space
- fix random seed for reproducibility

```
%%time !vw --oaa 10 -d
$PATH_TO_STACKOVERFLOW_DATA/stackoverflow_train.vw \ -f
vw_model1_10mln.vw -b 28 --random_seed 17 --quiet

# Output:

CPU times: user 559 ms, sys: 171 ms, total: 730 ms Wall
time: 38.5 s


%%time !vw -t -i vw_model1_10mln.vw -d
$PATH_TO_STACKOVERFLOW_DATA/stackoverflow_test.vw \ -p
vw_test_pred.csv --random_seed 17 --quiet

# Output:

CPU times: user 322 ms, sys: 97 ms, total: 420 ms Wall time:
22.8 s
```

```
vw_pred = np.loadtxt(os.path.join(PATH_TO_STACKOVERFLOW
                                  'vw_test_pred.csv'))
test_labels = np.loadtxt(os.path.join(PATH_TO_STACKOVER
                                      'stackoverflow_te
```

```
# Output:

0.91728604842865913
```

The model has trained itself and made predictions in less than a minute (check it, these results are reported for a MacBook Pro, mid 2015, 2.2 GHz Intel Core i7, 16GB RAM). Its accuracy is almost 92%. All of this without a Hadoop cluster! :) Impressive, isn't it?

4. Assignment #8

In this assignment, you'll practice yourself with Vowpal Wabbit and gigabytes of data. Fill the missing code in this Jupyter notebook, and then pick answers in a Google form. You can edit your responses even after submitting the form.

Hard deadline: April 14, 23:59 CET

5. Useful resources

- Official VW documentation on Github
- "Numeric Computation" Chapter of the Deep Learning book
- "Convex Optimization" by Stephen Boyd
- "Command-line Tools can be 235x Faster than your Hadoop Cluster" post
- Benchmarking various ML algorithms on Criteo 1TB dataset on GitHub
- VW on FastML.com

. . .

Author: Yury Kashnitsky. Translated and edited by Serge Oreshkov, and Yuanyuan Pao.