Maksym Shevchenko, AI-1

Laboratory work 2

Reuters-21578 text classification using SVM

Theory

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

Effective in high dimensional spaces.

Still effective in cases where number of dimensions is greater than the number of samples.

Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.

SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

More formally, a support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier.

SVMs are helpful in text and hypertext categorization, as their application can significantly reduce the need for labeled training instances in both the standard inductive and transductive settings.

More detailed information about SVM and equations you can find here: https://en.wikipedia.org/wiki/Support-vector_machine. Main formula of linear SVM is hyperplane equation $\mathbf{w}^T\mathbf{x} - b = 0$, where w is the (not necessarily normalized) normal vector to the hyperplane.

One-vs-rest (OvR for short, also referred to as One-vs-All or OvA) is a heuristic method for using binary classification algorithms for multi-class classification.

It involves splitting the multi-class dataset into multiple binary classification problems. A binary classifier is then trained on each binary classification problem and predictions are made using the model that is the most confident.

A possible downside of this approach is that it requires one model to be created for each class. For example, three classes requires three models. This could be an issue for large datasets (e.g. millions of rows), slow models (e.g. neural networks), or very large numbers of classes (e.g. hundreds of classes).

This approach requires that each model predicts a class membership probability or a probability-like score. The argmax of these scores (class index with the largest score) is then used to predict a class.

Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent. Conversely, stepping in the direction of the gradient will lead to a local maximum of that function; the procedure is then known as gradient ascent.

Instead of climbing up a hill, think of gradient descent as hiking down to the bottom of a valley. This is a better analogy because it is a minimization algorithm that minimizes a given function.

The equation below describes what gradient descent does: b is the next position of our climber, while a represents his current position. The minus sign refers to the minimization part of gradient descent. The gamma in the middle is a waiting factor and the gradient term (Δf(a)) is simply the direction of the steepest descent. $\mathbf{b} = \mathbf{a} - \gamma \nabla \mathbf{f}(\mathbf{a})$ this formula basically tells us the next position we need to go, which is the direction of the steepest descent.

A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

A vocabulary of known words.

A measure of the presence of known words.

Dataset

Reuters-21578 Text Categorization Collection Data Set. The collection consists of 22 data files, an SGML DTD file describing the data file format, and six files describing the categories used to index the data. Some additional files, which are not part of the collection but have been contributed by other researchers as useful resources are also included. All files are available uncompressed, and in addition a single gzipped Unix tar archive of the entire distribution is available as reuters21578.tar.gz.

The Reuters-21578 collection is distributed in 22 files. Each of the first 21 files (reut2-000.sgm through reut2-020.sgm) contain 1000 documents, while the last (reut2-021.sgm) contains 578 documents. The files are in SGML format. Readers interested in more detail on SGML are encouraged to pursue one of the many books and web pages on the subject.

Implementation

You can find source code here:
https://github.com/Gurdel/Magistracy/tree/main/Machine%20Learning/SVM.

1) Reuters dataset parsing and new dataset creation. Firstly, I placed all input files' text to one big file input.txt. Then I split text by tags combination <TOPIC><D> to get list of records with topic. In each record I found topic (text before <\D> tag) and message (text between <TITLE></TITLE> and between <BODY></BODY> tags). Since there were many different numbers in the messages, I replaced them with the special word int. After that, I normalized the text, leaving only the words separated by a space.

```
texts = inp.split('<TOPICS><D>')
for t in texts[1:]:
    try:
        topic = t.split('</D>')[0].replace('-', '')
        title = t.split('<TITLE>')[1]
        text = text.split('</TITLE>')[0]
        text = t.split('<BODY>')[1]
        text = ' ' + title + text.split('</BODY>')[0]

        #  Видаляємо (заміняємо на пробіл):
        #text = re.sub(r'(\d+[,.]?)+', 'int', text)  #  заміняємо числа на слово int
        text = re.sub('&lt', ' ', text)  #  &lt в .sgm
        text = re.sub('&#3;', ' ', text)  #  &#3; в .sgm
        text = re.sub(r'\W', ' ', text)  #  спец символи
        text = re.sub(r'\s+\D\s+', ' ', text)  #  одиничні літери
        text = re.sub(r'\s+\D\s+', ' ', text)
        text = re.sub(r'\s+\D\s+', ' ', text)
        text = re.sub(r'\^[a-zA-Z]\s+', ' ', text)  #  не літери
        text = re.sub(r'\s+[a-zA-Z]\$', ' ', text)
        text = re.sub(r'\s+', ' ', text, flags=re.I)  #  подвійні пробіли
        text = text.lower().strip()  #  переводимо в нижній регістр
        sw.write(f'{topic}\t{text}\n')
```

2) Vectorize text. I lemmatized text using nltk.stem.WordNetLemmatizer and vectorized it using bag-of-words (CountVectorizer from sklearn.feature_extraction.text). To decrease vectors dimension I decided to choose only 2000 most frequent words, set up they minimum frequency to 10 messages and maximum frequency to 80% of all messages. Also, I deleted English stopwords.

```
for t in X:
    lem = [wnl.lemmatize(word) for word in str(t).split()]
    lem_texts.append(' '.join(lem))

from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=2000, min_df=10, max_df=0.8,
                     stop_words=nltk.corpus.stopwords.words('english'))
#max_features - к-сть слів, які використовуються для класифікації
#min_df - мін к-сть текстів, у яких міститься слово
#max_df - макс відсоток файлів, у яких міститься слово
#stop_words - шумові слова
X = cv.fit_transform(lem_texts).toarray()
```

3) Classification. Let’s decide X it’s input matrix we got and y is target vector. Firstly, I scaled each column in X to range [0, 1], used OvA strategy to target vector (replace y[i] class number with vector of zeros and one 1 in index of class number), and initialize weight vector theta with zeros. I splited dataset to train and test sets and also split train set to train and validation test five times, to implement cross-validation. After that I run 50 iteration (for 5 validations, 250 epochs in total), in each of them for each class I applied gradient descend and changed class weights according to them.

```
# Gradient Descent on the weights/parameters
def gradDescent(theta, c, x, yT, learning_rate):
    oldTheta = theta[c]
    v = np.vectorize(sigmoid)
    a = v(np.sum(oldTheta * x, axis=1)) - yT
    for j in range(len(theta[c])):
        derivative_sum = a * x[:,j]
        theta[c][j] -= learning_rate*np.sum(derivative_sum)
```

```
def cross_validation(x, y, x_res, y_res, test_data_size, validations, learning_rate, epoch):
    print("Epochs count: ", epoch)
    accuracies = []
    theta = np.zeros((len(label_map), len(x[0])))
    for valid in range(validations):
        x_train, x_val, y_train, y_val = train_test_split(x, y,
                                                          test_size=test_data_size)
        # converting y_train to classwise columns with 0/1 values
        classes = np.transpose(y_train)
        # Initialising Theta (Weights)
        # training model
        for i in range(epoch):
            print(i, end='')
            for class_type in range(len(classes)):
                print('.', end='')
                gradDescent(theta, class_type, x_train,
                            classes[class_type], learning_rate)
```

4) After each validation for each sample I calculated probability of belonging to every class and choose max. After I compared model prediction with expected and got next values:

```
Validation 1 accuracy: 0.8437317784256559 Test data accuracy: 0.8526119402985075
```

```
Validation 2 accuracy: 0.9102040816326531 Test data accuracy: 0.8708022388059702
```

```
Validation 3 accuracy: 0.9469387755102041 Test data accuracy: 0.8740671641791045
```

```
Validation 4 accuracy: 0.9615160349854227 Test data accuracy: 0.8698694029850746
```

```
Validation 5 accuracy: 0.9749271137026239 Test data accuracy: 0.8722014925373134
```

5) To compare the results with an existing classifier, I used sklearn.svm.SVC classifier. As you can see, the accuracy of my classifier is quite good.

```
from sklearn import model_selection
from sklearn.svm import SVC
model = SVC(kernel='linear')

kfold = model_selection.KFold(n_splits=10)
cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold,
                                             scoring='accuracy')
msg = "%s: %f (%f)" % ('SVM', cv_results.mean(), cv_results.std())
print(msg)
```

```
SVM: 0.867490 (0.015561)
```

```
model.fit(X_train, y_train)
predictions = model.predict(X_test)
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
print(confusion_matrix(y_test,predictions))
```

```
0.8782649253731343
```

	precision	recall	f1-score	support
acq	0.86	0.97	0.91	459
alum	0.90	0.69	0.78	13
bop	1.00	0.36	0.53	11
carcass	0.50	0.33	0.40	6
cocoa	1.00	0.88	0.94	17
coconut	0.00	0.00	0.00	1
coffee	0.96	1.00	0.98	24
copper	0.88	0.93	0.90	15
corn	0.00	0.00	0.00	3
cotton	0.50	0.50	0.50	2
cpi	0.76	0.72	0.74	18
cpu	0.00	0.00	0.00	1
crude	0.90	0.84	0.87	95
dlr	0.00	0.00	0.00	5
earn	0.97	0.96	0.97	769
fcattle	0.00	0.00	0.00	2
fuel	0.00	0.00	0.00	1
gas	1.00	0.25	0.40	4
gnp	0.88	0.88	0.88	24
gold	0.92	0.96	0.94	24
grain	0.75	0.91	0.82	123
heat	0.00	0.00	0.00	2

You can also see the accuracy of the dataset classification by other classifiers:

SGDClassifier	0.8871268656716418
LinearSVC	0.8857276119402985
RidgeClassifier	0.8759328358208955
MLPClassifier	0.8703358208955224
SVC	0.8642723880597015
PassiveAggressiveClassifier	0.8638059701492538
LogisticRegression	0.8586753731343284
Perceptron	0.8521455223880597
RandomForestClassifier	0.8442164179104478
ensemble.ExtraTreesClassifier	0.84375
KNeighborsClassifier	0.8339552238805971
LabelPropagation	0.8288246268656716
BaggingClassifier	0.8283582089552238
ComplementNB	0.8190298507462687
GradientBoostingClassifier	0.8143656716417911
MultinomialNB	0.7779850746268657
DecisionTreeClassifier	0.7686567164179104
NearestCentroid	0.7467350746268657
BernoulliNB	0.7416044776119403
tree.ExtraTreeClassifier	0.6851679104477612
GaussianNB	0.6427238805970149
AdaBoostClassifier	0.4766791044776119
CategoricalNB	0.37406716417910446
DummyClassifier	0.18703358208955223