Introduction to Machine Learning Fall 2019 Homework 1
Cole Smith css459

Please view the code at:
`https://github.com/css459/intro-to-ml-hw1`

1. Question 1

   We need to create a validation set in order to assess the performance of our model on data it hasn't seen before, since we want to be able to apply the model on new data. This holdout set lets us simulate that situation of new data, except we know the answer and the model doesn't.

   If we had included the validation set into the training step, then the model would have an unfair advantage when scoring its performance on this set because it has already seen these points of data.

2. Question 2

   This is handled by the Email Class

```
# Constants
DATA_TRAIN_FILE = "../data/spam_train.txt"
DATA_TEST_FILE = "../data/spam_test.txt"


class Email:

    def __init__(self, tokens, is_spam):
        self.tokens = tokens
        self.is_spam = int(is_spam) == 1

    def vectorize_tokens(self, vocabulary):
        """
        Given a vector of text information in the `tokens` field,
        transform this into a binary word-occurrence vector for all
        words in the vocabulary.

        :param vocabulary: A list of words for which this output
                           vector will represent. The output vector
                           will be the same length as this vector,
                           and the order of the vocabulary vector
                           will determine the order of this output
                           vector.

        :return: An ordered binary vector of length `vocabulary`.
        """
        vec = []
        words = set(self.tokens)
        for v in vocabulary:
            if v in words:
                vec.append(1)
            else:
                vec.append(0)

        return vec
```

```
@staticmethod
def load_emails_from_data(validation_percent=0.20):
    with open(DATA_TRAIN_FILE, "r") as fp:
        emails = []

        # Parse each line in the file into email tokens
        # and Y variable
        for line in fp:
            is_spam = int(line.split()[0])
            tokens = line.split()[1:]

            new_email = Email(tokens, is_spam)
            emails.append(new_email)

        print("[␣INF␣]␣Read", len(emails), "samples.")
        print("[␣INF␣]␣Train␣count:", len(emails) * (1 -
            red↪ validation_percent),
              "Test␣Count:", len(emails) * validation_percent)

        split_index = int(len(emails) * (1 - validation_percent))
        return emails[:split_index], emails[split_index:]

@staticmethod
def load_test_file():
    with open(DATA_TEST_FILE, "r") as fp:
        emails = []

        # Parse each line in the file into email tokens
        # and Y variable
        for line in fp:
            is_spam = int(line.split()[0])
            tokens = line.split()[1:]

            new_email = Email(tokens, is_spam)
            emails.append(new_email)

        print("[␣INF␣]␣Read", len(emails), "samples.")
```

```
            print("[␣INF␣]␣Count:", len(emails))

            return emails
```

3. Question 3

This is handled by the PerceptronClassifier Class

```
class PerceptronClassifier:

    def __init__(self, threshold=0.5, learning_rate=0.1, max_iter=100):
        """
        Simple Perceptron classifier. The threshold of the
        output decision can be set based on the class values
        in Y, and the learning rate can be adjusted. The
        weights will update row-by-row in order of the data
        X.

        :param threshold: Threshold activation for the Perceptron.
        :param learning_rate: Influence given to new weights over
                              previous weights.
        :param max_iter: Maximum iterations for training.
        """
        # The weights provided by the train() function
        self.weights = None

        # The length of the weights array (dimensionality of data)
        self.dimensionality = None

        self.features = None

        self.threshold = threshold
        self.learning_rate = learning_rate
        self.max_iter = max_iter

    def _check_inputs(self, x, y):
        if x is None or len(x) == 0:
            print("[␣ERR␣]␣No␣data␣provided:␣Either␣empty␣set␣or␣None␣
                red↪ was␣provided.")
            return None, None

        if len(x) != len(y):
            print("[␣ERR␣]␣X␣and␣Y␣are␣not␣the␣same␣length")
```

```
            return None, None

        if isinstance(x, pd.DataFrame):
            if not self.features:
                self.features = x.columns.tolist()

            x = x.values

        if isinstance(y, pd.DataFrame):
            y = y.values

        return x, y

    def _is_training_error(self, x, y, w):
        errors = 0
        pred = [int(np.dot(x_i, w) >= self.threshold) for x_i in x]
        for i in range(len(pred)):
            if pred[i] != y[i]:
                errors += 1

        print("\tTraining␣error:", errors)
        return errors > 0

    def fit(self, x, y):
        """
        Fits the classifier on the X, Y training data set.
        If X and Y are not linearly separable, the weight
        output will not be meaningful.
        Sets internal property, `self.weights`.

        :param x: Training data. Can be either 2D array or DataFrame
                  If a DataFrame is provided `self.features` will be
                  populated with the columns in X.
        :param y: Labels for training data X.
        :return: `None` Sets internal property `self.weights`.
        """

        # Validate inputs
```

```
        x, y = self._check_inputs(x, y)
        if x is None or y is None:
            return

        print("[␣INF␣]␣Fitting␣Perceptron␣with", len(x[0]), "
            red↪ dimensionality")

        # The weights array, in dimensionality of data
        # The initial values will be zero
        w = len(x[0]) * [0]

        for i in range(self.max_iter):
            print("\tIter:", i)
            for j in range(len(x)):
                # X and Y for the jth sample
                x_j = x[j]
                d_j = y[j]

                # The model output at time t for sample j
                y_jt = int(np.dot(w, x_j) >= self.threshold)

                if d_j - y_jt == 0:
                    continue

                # The new weight vector for t + 1
                w = [w[i] + (0.01 * (d_j - y_jt) * x_j[i]) for i in
                    red↪ range(len(w))]

            if not self._is_training_error(x, y, w):
                break

        # Assign the final weights to this object instance
        self.weights = w
        self.dimensionality = len(x[0])

    def predict(self, x, weights=None):
        """
        Make predictions Y on provided X.
```

```
        :param x: Unlabelled input data. Can be
                          either DataFrame or 2D array.
        :param weights: Optional weights vector.
                          `self.weights` used by default.
        :return: Labels vector Y for X.
        """
        y = []

        if weights is None:
            weights = self.weights

        if isinstance(x, pd.DataFrame):
            x = x.values

        for x_i in x:
            y.append(int(np.dot(x_i, weights) >= self.threshold))

        return y

    def validate(self, val_x, val_y):
        """
        Prints the F1 and Confusion Matrix for
        classifier.

        :param val_x: Validation data X.
        :param val_y: Validation data Y.
        :return: `None`
        """
        if self.weights is None:
            print("[ ERR ] Could not validate model: Not fitted")
            return

        val_x, val_y = self._check_inputs(val_x, val_y)
        if val_x is None or val_y is None:
            return

        y_pred = self.predict(val_x)
```

```
        print("F1␣Score:", f1_score(val_y, y_pred))
        print(confusion_matrix(val_y, y_pred))

    def save_weights(self, filename='perceptron_weights.json'):
        if self.weights is None:
            print("[␣ERR␣]␣Cannot␣save␣weights:␣Not␣fitted")
            return

        with open(filename, 'w') as fp:
            json.dump(self.weights, fp, indent=4)

    def save_features(self, filename='feature_weights.csv'):
        if self.weights is None or self.features is None:
            print("[␣ERR␣]␣Cannot␣save␣features:␣Not␣fitted␣or␣no␣
                red↪ features")
            return

        df = pd.DataFrame()
        df['feature'] = self.features
        df['weight'] = self.weights
        df.to_csv(filename, index=False)

    def load_weights(self, filename='doc/perceptron_weights.json'):
        with open(filename, 'r') as fp:
            w = json.load(fp)
            self.weights = w
```

And implemented in `main.py`

```
from src.model.email import Email
from src.model.perceptron_classifier import *
from src.preprocess import get_vocabulary_vector


def create_pandas_dataframes():
    """
    Automatic function to form train and test
    Pandas DataFrames.
```

```
    :return: Train and Test set, respectively
    """
    train, test = Email.load_emails_from_data()

    train_y = [int(t.is_spam) for t in train]
    test_y = [int(t.is_spam) for t in test]

    vocab = get_vocabulary_vector(train)
    print("[␣INF␣]␣Vocab␣Size:", len(vocab))

    train = [t.vectorize_tokens(vocab) for t in train]
    test = [t.vectorize_tokens(vocab) for t in test]

    train = pd.DataFrame.from_records(train, columns=vocab)
    test = pd.DataFrame.from_records(test, columns=vocab)

    train['is_spam'] = train_y
    test['is_spam'] = test_y

    return train, test


def perceptron_train(train_df):
    train_x = train_df.drop('is_spam', 1)
    train_y = train_df['is_spam']

    p = AveragedPerceptronClassifier()
    p.fit(train_x, train_y)

    p.save_features()
    p.save_weights()

    return p.weights


def perceptron_test(w, test_df):
    test_x = test_df.drop('is_spam', 1)
    test_y = test_df['is_spam']
```

```
    p = AveragedPerceptronClassifier()
    p.weights = w
    p.validate(test_x, test_y)


def train_test():
    train_df, test_df = create_pandas_dataframes()
    train_x = train_df.drop('is_spam', 1)
    train_y = train_df['is_spam']
    test_x = test_df.drop('is_spam', 1)
    test_y = test_df['is_spam']

    p = AveragedPerceptronClassifier(max_iter=50)
    p.fit(train_x, train_y)
    p.validate(test_x, test_y)


def train_final():
    train_df, test_df = create_pandas_dataframes()
    final = pd.concat([train_df, test_df], ignore_index=True)

    final_x = final.drop('is_spam', 1)
    final_y = final['is_spam']

    p = PerceptronClassifier(max_iter=10)
    p.fit(final_x, final_y)

    p.save_weights()
    p.save_features()

    vocab = final_x.columns
    test = Email.load_test_file()
    test_y = [int(t.is_spam) for t in test]
    test = [t.vectorize_tokens(vocab) for t in test]

    test = pd.DataFrame.from_records(test, columns=vocab)
    test['is_spam'] = test_y
```

```
    test_x = test_df.drop('is_spam', 1)
    test_y = test_df['is_spam']

    p.validate(test_x, test_y)


if __name__ == '__main__':
    # This is only used to make to consistent with
    # what the homework asks. The way I use my Perceptron
    # is in `train_test()`:
    # train_df, test_df = create_pandas_dataframes()
    # perceptron_test(perceptron_train(train_df), test_df)

    # train_test()

    # Final, full training on all data
    train_final()
```

4. Question 4

   The algorithm times out at 100 iterations, with a training error of 1 data point. The validation F1 score is 0.94 with 33 misclassified points.

5. Question 5

   These were determined using the final best perceptron on all of the `spam_train` file, and validated on the `spam_test` file. The vocabulary was created using a 30-frequency threshold will all eligible words from all of `spam_train`.

   The following are the top most positive by weight:

```
click 0.27
number 0.25
sight 0.24
pleas 0.23
basenumb 0.21
here 0.19
your 0.19
deathtospamdeathtospamdeathtospam 0.19
exit 0.19
httpaddr 0.18
remov 0.18
guarante 0.18
instruct 0.18
form 0.17
nbsp 0.16
```

The following are the top most negative by weight:

```
wrote -0.23
inc -0.16
review -0.16
i -0.15
but -0.15
prefer -0.15
on -0.14
recipi -0.14
from -0.13
and -0.13
spam -0.13
yahoo -0.13
version -0.13
plain -0.13
technolog -0.13
```

6. Question 6

```
class AveragedPerceptronClassifier(PerceptronClassifier):
    def __init__(self, threshold=0.5, learning_rate=0.1, max_iter=100):
        PerceptronClassifier.__init__(self,
                                      threshold=threshold,
                                      learning_rate=learning_rate,
                                      max_iter=max_iter)

    def fit(self, x, y):
        """
        Fits the classifier on the X, Y training data set.
        If X and Y are not linearly separable, the weight
        output will not be meaningful.
        Sets internal property, 'self.weights'.
        Final weight vector is the average of all considered
        weight vectors.

        :param x: Training data. Can be either 2D array or DataFrame
                  If a DataFrame is provided 'self.features' will be
                  populated with the columns in X.
        :param y: Labels for training data X.
        :return: 'None' Sets internal property 'self.weights'.
        """

        # Validate inputs
        x, y = self._check_inputs(x, y)
        if x is None or y is None:
            return

        print("[␣INF␣]␣Fitting␣Perceptron␣with", len(x[0]), "
            red↪ dimensionality")

        # The weights array, in dimensionality of data
        # The initial values will be zero
        w = len(x[0]) * [0]

        # Average accumulator
```

```
        acc = w
        count = 0

        for i in range(self.max_iter):
            print("\tIter:", i)
            for j in range(len(x)):
                # X and Y for the jth sample
                x_j = x[j]
                d_j = y[j]

                # The model output at time t for sample j
                y_jt = int(np.dot(w, x_j) >= self.threshold)

                if d_j - y_jt == 0:
                    continue

                # The new weight vector for t + 1
                w = [w[i] + (0.01 * (d_j - y_jt) * x_j[i]) for i in
                   red↪ range(len(w))]
                acc = [acc[i] + w[i] for i in range(len(acc))]
                count += 1

            if not self._is_training_error(x, y, w):
                break

        # Assign the final weights to this object instance
        self.weights = [x / count for x in acc]
        self.dimensionality = len(x[0])
```

7. Question 7

   See `PerceptronClassifier.max_iter`

8. Question 8

   Of 10, 30, 50, and 100 iterations, the best I found for each was:

   Regular: 10 Iterations, F1: 0.96

```
10
F1 Score: 0.96
[[675  10]
 [ 15 300]]

30
F1 Score: 0.9559748427672955
[[668  17]
 [ 11 304]]

50
F1 Score: 0.9543307086614173
[[668  17]
 [ 12 303]]

100
(below)
```

Averaged: 100 Iterations, F1:0.969

```
10
F1 Score: 0.9538950715421304
[[671  14]
 [ 15 300]]

30
F1 Score: 0.9555555555555556
[[671  14]
 [ 14 301]]

50
F1 Score: 0.9587301587301588
[[672  13]
 [ 13 302]]

100
(below)
```

9. Question 9

   All training data, Regular Perceptron:

   Using tunings from best Perceptron above. Trained on 5000 samples, tested on 1000 samples from testing file. Vocab from all of `spam_train` file.

```
Max Iterations: 10
Training Error: 3

F1 Score: 0.9952153110047847
[[685   0]
 [  3 312]]
```

   (Honorable Mention: Averaged Perceptron, 100 Iterations)

```
F1 Score: 0.969502407704655
[[679   6]
 [ 13 302]]
```

1 Results

```
Regular Preceptron after 100 iterations with 1 training error:

[ INF ] Read 5000 samples.
[ INF ] Train count: 4000.0 Test Count: 1000.0
[ INF ] Vocab Size: 2376
[ INF ] Fitting Perceptron with 2376 dimensionality

F1 Score: 0.9473684210526315
[[670  15]
 [ 18 297]]

-------------------------

Averaged Perceptron after 100 iterations with 1 training error:

[ INF ] Read 5000 samples.
[ INF ] Train count: 4000.0 Test Count: 1000.0
[ INF ] Vocab Size: 2376
[ INF ] Fitting Perceptron with 2376 dimensionality

F1 Score: 0.9570747217806042
[[672  13]
 [ 14 301]]
```

```
-------------------------
-------------------------
Final result

All training data, Regular Perceptron:

Using tunings from best Perceptron above.
Trained on 5000 samples, tested on 1000 samples from testing file.
Vocab from all of spam_train file.

Max Iterations: 10
Training Error: 3

F1 Score: 0.9952153110047847
[[685   0]
 [  3 312]]

(Honorable Mention: Averaged Perceptron, 100 Iterations)
F1 Score: 0.969502407704655
[[679   6]
 [ 13 302]]
```