hw4

June 1, 2020

1 Satya Phani Rahul Damineni (933-922-122)

1.1 Need for pre-processing data

1.1.1 All words are lower-cased

- For sentiment classification task, casing of word may not matter significantly to determine sentiment of the sentence
- So, by converting everything to an uniform case, we are making the modeling task easy by telling the algorithm that either case is semantically same

1.1.2 Punctuation became separate words

- The first step for NLP is to convert word symbols into numerical vectors so that modeling would be possible
- If we leave punctuations with the adjoining word, it will be interpreted as a different token (word) – this is semantically inconsistent

1.1.3 Verb contractions are split into their component morphemes

- This is data augmentation step too. We want our model to see you'll as you + 'll so that it can use its understanding (obtained during training) about "you" and "'ll" than seeing it as a new word and eventually deriving this rule (if at all) during training.
- In general, all fixed rules (syntactic) about the data should be made aware to the model to make its job of deriving patterns easy. In this case, we are imposing such rules on the data and making sure the model never have to learn them while training.

1.1.4 Quotes are re-written

- Same as above.
- As long as you consistently map quotes to a new (rare) character, you don't exactly need to use forward & backward quotes. The model should be able to make out a pattern from the data it was given.

1.1.5 Spanish reviews were removed.

- Since the quantity is small, there will be conflicting symbols to the same semantic token – this can slow down the learning algorithm if not throws it off.

1.2 Naive Perceptron Baseline

1.2.1 Understanding `svector.py`

- For each operation, we are updating values of the `svectors` data-structure `defaultdict`.
- We do that by accessing `key`, `value` pairs of `defaultdict` and updating `value` based on the required operation (`add`, `subtract`, `scalar_multiplication`, `dot_product`) and the second arguement `b`. We retrive `value` of `key` from `b` using the `key` from the original (`self`) `original [operation] b[key]`
- The advantange in using `defaultdict` appears when `key` value gets a default `0` instead of raising `KeyError` when it is not present in `b`

1.2.2 Understanding `train.py`

- `train` initializes `W` and runs for `epoch` number of times
- In each epoch, it retrives "train" raw words and their labels iteratively, converts words into a `svector`.
- It then checks if the dot product of retrieved sentence and W yields correct label and updates `W` if it doesn't
- It also evaluates the dev error and keeps track of best error for each epoch
- `test` retrieves "dev" raw words and their labels and uses the trained W to determine how many errors happen
- It returns the normalized error of the entire dev set for the current `W`
- The epoch summaries and final summary would be printed

1.2.3 Adding bias term

- Yes, adding bias reduced to dev error to `26.3%`
- I added bias by adding an extra dimension (`--bias--`) and set it to `1` to all sentence vectors.

1.2.4 Why is bias helping on a balanced dataset?

- Sometimes a dataset may not be linearly seperable through origin – bias term accommodates this scenario
- Even though the dataset is balanced, depending upon the order of training data, the final model vector could be slightly misaligned causing the nearby points to be misclassified. The bias term compensates by giving slight translation (`bias = -2.0`) to avoid this – based on the order of input examples.

1.3 Averaged Perceptron

1.3.1 Accuracy: `26.3%`. Averaging made dev errors consistently smooth

1.3.2 Training speed: It slightly slowed down training (to `1.9s`) owing to extra ops in calculating averaged model

1.3.3 Feature importances:

20 most negative features:

```
[('boring', -1193063.0), ('generic', -1044776.0), ('dull', -1030705.0), ('badly', -949415.0),
32.0), ('ill', -880309.0), ('too', -846307.0), ('instead', -817796.0), ('tv', -816025.0), ('at
.0), ('incoherent', -788491.0), ('neither', -787754.0), ('flat', -782628.0), ('seagal', -77299
ed', -767592.0), ('worst', -766857.0), ('suffers', -765722.0)]
```

20 most positive features:

```
[('flaws', 750394.0), ('smarter', 750566.0), ('imax', 768587.0), ('delightful', 778127.0), ('p

, ('refreshingly', 804340.0), ('wonderful', 813079.0), ('dots', 816579.0), ('cinema', 819089.0
015.0), ('treat', 847015.0), ('skin', 849652.0), ('french', 864314.0), ('provides', 878381.0),
237.0), ('triumph', 906538.0), ('engrossing', 975282.0)]
```

Yes, most of them make sense. But `flaws` being the most positive feature is surprising. `Segal` seem to be a bad director, I don't recognise any of this movies.

1.3.4 Top 5 false positives and false negatives:

False positives:

```
["the thing about guys like evans is this you 're never quite sure where self promotion ends a
 watch the movie , you 're too interested to care", 'neither the funniest film that eddie murp
howtime is nevertheless efficiently amusing for a good while before it collapses into exactly
 to lampoon , anyway', "even before it builds up to its insanely staged ballroom scene , in wh
, it 's waltzed itself into the art film pantheon", "if i have to choose between gorgeous anim
 time", 'carrying off a spot on scottish burr , duvall ( also a producer ) peels layers from t
isted on paper']
```

False negatives:

```
['an atonal estrogen opera that demonizes feminism while gifting the most sympathetic male of
vomit bath at his wedding', 'mr wollter and ms seldhal give strong and convincing performances
 recesses of the character to unearth the quaking essence of passion , grief and fear', 'bravo
 carefully selecting interview subjects who will construct a portrait of castro so predominant
nd likable you find them", "` in this poor remake of such a well loved classic , parker expose
basic flaws in his vision '"]
```

These make sense too: `nevertheless efficiently amusing for a good while before it collapses into exactly the kind of buddy cop comedy it set` is a hard one to classify by assigning weights to features without realizing their context.

1.3.5 Caching: imroved speed to 1.7s. This was effective. I didn't optimize what I'd be making a hash key, otherwise could have improved further.

1.4 Pruning the Vocabulary

1.4.1 Neglecting one-count words: dev error = 25.9%. It improved – model regularized

1.4.2 Update % = 10.6% at 10th epoch: Meaning model is generalizing (towards underfitting than over fitting)

1.4.3 Model size = 8425 (Yes, almost halved)

1.4.4 Training speed = 2.1s (without caching). Not much, I think computing dot product isn't the bottleneck.

1.4.5 If I neglect <=2 word counts: dev error = 26.6%, for <=3 word counts: dev error = 27%. Model size is shrinking though. Even thought it didn't improve, notice that it didn't cause as much damage.

1.5 Other learning algorithms

[68]:
```
import json
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.feature_extraction.text import TfidfVectorizer
```

[2]:
```
def load_data(textfile):
    '''
    Loads and returns input sentences and their labels (+ / -)
    '''
    with open(textfile) as in_:
        examples = in_.readlines()

    X, y = [], []
    for ex in examples:
        label, sent = ex.strip().split("\t")
#         label = 1 if label == "+" else -1

        X.append(sent)
        y.append(label)

    return X, y
```

[7]:
```
train_X, train_y = load_data("hw4-data/train.txt")
dev_X, dev_y = load_data("hw4-data/dev.txt")
test_X, _ = load_data("hw4-data/test.txt")
```

```
[12]: stop_words = set()
      with open("hw4-data/tokens.json") as in_:
          pruned_tokens = json.load(in_)

      tokens_index = {tk: i for i, tk in enumerate(pruned_tokens)}
```

```
[15]: def input_normalizer(list_of_sent):

          out = np.zeros((len(list_of_sent), len(tokens_index)))
          for sid, sent in enumerate(list_of_sent):
              tokens = sent.split()

              for token in tokens:
                  try:
                      out[sid, tokens_index[token]] += 1
                  except KeyError:
                      continue

          return out
```

```
[23]: tr_X = input_normalizer(train_X)
      dv_X = input_normalizer(dev_X)
      ts_X = input_normalizer(test_X)
```

```
[26]: clf = GradientBoostingClassifier(random_state=0)
      clf.fit(tr_X, train_y)
```

```
[26]: GradientBoostingClassifier(criterion='friedman_mse', init=None,
                    learning_rate=0.1, loss='deviance', max_depth=3,
                    max_features=None, max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=100,
                    n_iter_no_change=None, presort='auto', random_state=0,
                    subsample=1.0, tol=0.0001, validation_fraction=0.1,
                    verbose=0, warm_start=False)
```

```
[33]: print(f'Dev error: {(1 - clf.score(dv_X, dev_y)) * 100:.2f}%')
```

```
Dev error: 35.60%
```

- I quickly tried gradient boosting and it performed much worse than Avg. Perceptron.
- The main reason for this is how the weak classifiers could have very little/ no info about the polarity all possible features that they could encounter while they are trained only on a small subset of data.
- The strength of this algorithm is how well it can regularise but in this case it's just throwing the data away!

```
[118]: corpus = [
               open("hw4-data/train.txt").read(),
                   open("hw4-data/dev.txt").read(),
                       open("hw4-data/test.txt").read()
       ]

       word_vectorizer = TfidfVectorizer(
           sublinear_tf=True,
           analyzer='word',
           stop_words='english',
           ngram_range=(1, 10),
           max_features=20000)
       word_vectorizer.fit(corpus)
```

```
[118]: TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
               dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
               lowercase=True, max_df=1.0, max_features=20000, min_df=1,
               ngram_range=(1, 10), norm='l2', preprocessor=None, smooth_idf=True,
               stop_words='english', strip_accents=None, sublinear_tf=True,
               token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
               vocabulary=None)
```

```
[119]: tr_X = word_vectorizer.transform(train_X)
       dv_X = word_vectorizer.transform(dev_X)
       ts_X = word_vectorizer.transform(test_X)
```

```
 [60]: gbc = GradientBoostingClassifier()
       gbc.fit(tr_X, train_y)
```

```
 [60]: GradientBoostingClassifier(criterion='friedman_mse', init=None,
                     learning_rate=0.1, loss='deviance', max_depth=3,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_iter_no_change=None, presort='auto', random_state=None,
                     subsample=1.0, tol=0.0001, validation_fraction=0.1,
                     verbose=0, warm_start=False)
```

```
 [67]: print(f'Dev error (Gradient Boosting Classifier): {(1 - gbc.score(dv_X, dev_y))␣
        ↪* 100:.2f}%')
```

```
Dev error (Gradient Boosting Classifier): 39.30%
```

```
 [64]: lr = LogisticRegression()
       lr.fit(tr_X, train_y)
```

```
/Users/dsp/Library/Python/3.7/lib/python/site-
```

```
packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
[64]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                intercept_scaling=1, max_iter=100, multi_class='warn',
                n_jobs=None, penalty='l2', random_state=None, solver='warn',
                tol=0.0001, verbose=0, warm_start=False)
```

```
[66]: print(f'Dev error (Logistic Regression): {(1 - lr.score(dv_X, dev_y)) * 100:.
       ↪2f}%')
```

```
Dev error (Logistic Regression): 27.00%
```

```
[71]: lrcv = LogisticRegressionCV(cv=5, max_iter=500).fit(tr_X, train_y)
```

```
[76]: print(f'Dev error (Logistic Regression): {(1 - lrcv.score(dv_X, dev_y)) * 100:.
       ↪2f}%')
```

```
Dev error (Logistic Regression): 26.50%
```

1.6 Deployment

- Logistic Regression
- Dev error = 25.7%
- Params = {'C': 4.281332398719396, 'penalty': 'l2', 'solver': 'liblinear'}

```
[120]: param_grid = {
           'penalty' : ['l1', 'l2'],
           'C' : np.logspace(-4, 4, 20),
           'solver' : ['liblinear']
       }
       clf = GridSearchCV(LogisticRegression(), param_grid = param_grid, cv = 5,␣
        ↪verbose=True, n_jobs=-1)
       best_clf = clf.fit(tr_X, train_y)
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:    2.5s finished
```

```
[121]: print(f'Dev error (Logistic Regression): {(1 - best_clf.score(dv_X, dev_y)) *␣
        ↪100:.2f}%')
```

```
Dev error (Logistic Regression): 25.70%
```

```
[122]: best_clf.best_params_
```

```
[122]: {'C': 4.281332398719396, 'penalty': 'l2', 'solver': 'liblinear'}
```

[128]:
```
predictions = best_clf.predict(ts_X).tolist()
output = []
for sent, pred in zip(*[test_X, predictions]):
    output.append(pred + "\t" + sent)

with open("predictions.y", "w+") as out:
    out.write("\n".join(output))
```

[97]:
```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import SGDClassifier
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
```

[81]:
```
sgdc = make_pipeline(
        StandardScaler(with_mean=False),
        SGDClassifier(max_iter=1000, tol=1e-3)
).fit(tr_X, train_y)
print(f'Dev error (SGD Regression): {(1 - sgdc.score(dv_X, dev_y)) * 100:.2f}%')
```

```
Dev error (SGD Regression): 33.30%
```

[83]:
```
svmc = make_pipeline(StandardScaler(with_mean=False),
                     LinearSVC(random_state=0, tol=1e-5))
svmc.fit(tr_X, train_y)
print(f'Dev error (SVM Classifer): {(1 - svmc.score(dv_X, dev_y)) * 100:.2f}%')
```

```
Dev error (SVM Classifer): 30.20%
```

```
/Users/dsp/Library/Python/3.7/lib/python/site-packages/sklearn/svm/base.py:931:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

1.7 Debriefing

1. 6 hours
2. Moderate
3. Alone
4. 80%
5.

[ ]:
```
```