SVM

Accuracy :
The models accuracy as a whole in predicting tags from input sentences. Trained on 100 sentences (equivalent to 2019 word vectors) and tested on 1000 test sentences (equivalent to 20091 words) (time to train model 5-6 mins)

	Our accuracy	sklearn's SVC accuracy
1st fold	83.16 %	89.97 %
2nd fold	85.55 %	89.75 %
3rd fold	86.26 %	90.34 %
4th fold	85.90 %	90.11 %
5th fold	85.23 %	89.52 %
Avg accuracy	85.25 %	~ 90 %

Confusion Matrix :
Averaged across all tags for every fold. (Total - 20091 in test dataset) (For an average one vs rest tag classifier)

1584	274
272	18312

per-POS accuracy :
Averaged across every fold

.	99.99 %
NOUN	93.37 %
VERB	96.49 %
NUM	92.44 %
DET	99.35 %
PRON	99.5 %
PRT	98.94 %
ADP	97.93 %
X	0 %
ADV	97.52 %
ADJ	95.16 %
CONJ	99.87 %

Strengths of using SVMs for POS tagging:

Since we are using Linear SVM for POS tagging, the feature vectors need to be quite robust in order for the algorithm to converge under relatively tight margin bounds.

Thus for training feature vectors of size approx 450-500, usually suffices to get an average accuracy above 85%. The strengths with this method lies in the fact that if the feature vectors are not correct (i.e. incorrectly chose features) then the accuracy will be very low. But if features are chosen properly then even a small subset (approx 200 sentences) will result in an accuracy of around 90% on a test dataset of 10000 sentences.

This is quite useful since for languages for which large corpus is not available SVMs show just capturing the essential features from the language will give quite good results.

Weakness witnessed for SVM for POS tagging:

One primary weakness of using Linear SVM for POS tagging lies in the fact that if we are unable to design the feature vectors correctly the impact of this on the algorithm could be huge since it tries to classify data using a hyperplane (which can be relaxed to some extent by using a soft margin classifier or using kernels).

Another weakness is the time required to train the model. Since we are solving a quadratic optimization problem here, scaling with an increasing number of examples for training is not feasible. (There are alternatives to directly solving the quadratic optimization problem by using an algorithm called SMO which solves the dual version of the problem, allowing for kernels also, but we left it due to its complicated implementation.)
To train 2000 examples, we need to solve an optimization problem involving a 2000x2000 matrix with each vector and that too for each tag individually since it is a multi classification problem.
This makes the training too time consuming for this model.

Error Analysis:

- Lower accuracy than sklearn.svm's SVC since they use probabilistic weights to choose the best tag class among all valid tags.
- Increasing the training dataset size only increases the time taken to train the model without any notable increase in accuracy.
- Due to small dataset rare tag words of 'X' are not seen leading to a zero probability for that tag class.
- Prefix and suffix play an important role as features, hugely impacting the accuracy of the model.
- Some error creeps in due to the binary scheme of classification, which does not appear in sklearn's implementation.
- Using word embeddings only poorly increases the accuracy. Using a one-hot encoding of words hugely increases the accuracy (above 90 %) but leads to an extremely large feature vector (of size of 10K+).
- Using an optimal regularization (for soft margins) of '2' gives the best results. Values below 1 and above 5 decrease accuracy.

Learnings:

1. Theoretical & mathematical background behind SVM.

2. Converting the SVM problem to its dual version and finding a solution for that.
3. Sequential Minimal Optimization algorithm
4. Pegasos algorithm using SGD for solving SVM problem
5. Using cvxopt library to get the solution to the SVM problem
6. Calculate word embeddings using word2vec
7. Word stemming algorithms like Snowball stemmer
8. Efficient feature extraction from english words

References:

1. https://www.nltk.org/book/ch02.html
2. http://cs229.stanford.edu/notes/cs229-notes3.pdf - SVM
3. http://cs229.stanford.edu/materials/smo.pdf - SMO
4. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
5. https://people.csail.mit.edu/dsontag/courses/ml16/slides/lecture6_notes.pdf - Pegasos Algorithm
6. https://xavierbourretsicotte.github.io/SVM_implementation.html - Using cvxopt
7. https://rare-technologies.com/word2vec-tutorial/