Data Mining: Learning from Large Data Sets - Fall Semester 2015

mwurm@student.ethz.ch
merkim@student.ethz.ch
lwoodtli@student.ethz.ch

November 7, 2015

Large Scale Image Classification

Problem Description

The goal of the project was to develop a program that classifies images based on their visual content. The images were provided as features in a text file that has been extracted in a preprocessing step. Our task was to train a model that performs well given the feature representation by using support vector machines and parallel stochastic gradient descent in a map reduce environment.

Approach of the Team

The team decided that each member will develop the algorithm independently in a first step. The result of this first step was one mapper which was using the SGDClassifier from the scikit learn package and two mappers with an own implementation of the stochastic gradient descent approach. All of the mappers were able to reach an accuracy around 0.75. In order to improve the score the team decided to implement PEGASOS as well as AdaGrad. Unfortunately with both implementations the accuracy could not be significantly improved which showed us that we need a transformation in order to make the data separable. We therefore tried various forms of transformations of the data. Approximating kernels with random fourier features turned out to be the best choice for our data.

Environment

The running environment with a map reduce framework was provided. So the mapper and reducer were developed independently and then submitted to the running environment.

Map

Our final mapper uses a data transformation step to approximate kernels with random fourier features.

```
np.random.seed(42)
omegas = np.random.multivariate_normal(
                np.zeros(DIMENSION), 300*np.eye(DIMENSION),D)
bs = 2.*np.pi*np.random.random(D)
# precalculate for performance issues
prefactor = np.sqrt(2)
def transform(x_original):
        return prefactor * np.array(map( lambda omega,
                b: np.cos(np.dot(omega,x_original)+b) ,omegas,bs))
```

After the transformation the stochastic gradient descent implementation of scikit learn is used in order to find an optimal weight vector w. This is done for batches of size 100.

```
# create and use classifier and transformer
clf = SGDClassifier(loss="hinge", alpha=0.0000001,
                        fit_intercept=False, average=True)

# read and format data
currentBatch = ""
for i, line in enumerate(sys.stdin):
        if i % BATCHSIZE == 0 and i != 0:
            # prepare current batch
                data = np.fromstring(currentBatch, sep=' ')
                data = np.reshape(data,(-1,DIMENSION+1))
                Y = data[:, 0]
                X = data[:, 1:DIMENSION+1]

                # Do the transformation for each x
                x_transformed = np.array([transform(x) for x in X])

                # fit current batch partially
                clf.partial_fit(x_transformed, Y, classes=np.array(CLASSES))

                # clear batch
                currentBatch = ""
        currentBatch += line
```

At the end the resulting weight vector w is sent to the reducer.

Reduce

The reducer just calculates the average of the weight vectors from the different mappers.

Improvement

???

Conclusion

This project shows that linear classifiers like support vector machines can also be used for data which is not linearly separable. But in order to do this a transformation step is needed. The data needs to be transformed into a high-dimensional space where the data is separable. Doing this based on random fourier features turned out to be a good approach for the data provided for this project.