Machine Learning Engineer Nanodegree

Capstone Project

Elena Bushmanova
September and October, 2019

I. Definition

Project Overview

Peptidic natural products (PNPs) are small bioactive compounds consisting of amino acids connected via peptide bonds. A PNP may be represented as a graph with amino acids as nodes and bonds as edges. These graphs have either linear, cyclic, or more complex structure. PNPs are important for medicine since many of them are active against bacteria i.e. could be **antibiotics**. One of the main ways to study PNPs is through mass spectrometry. For each PNP you can get a **spectrum** (intensity as a function of the mass-to-charge ratio) or a few by examining it in a black box -- mass spectrometer. These spectra can further be compared against databases of previously characterized compounds using computational methods such as **DEREPLICATOR** (Mohimani H. et al., 2017).

Understanding which spectra correspond to which types of PNPs structure will significantly speed up the DEREPLICATOR since it will be possible to search through smaller sets (cyclic spectra only against cyclic compounds and linear only against linear). At the same time it will increase precision of the algorithm because initial DEREPLICATOR compares any spectrum with any compound and thereby can get such false positive matching as linear spectrum to non-linear compound and non-linear spectra to linear compound (not present in an improved algorithm). Also knowledge about the structure itself (separately from DEREPLICATOR) tells scientists some biological properties of the compound represented by its own spectrum. Cyclic PNPs are more stable and biologically more active on average so we can focus on studying of only such spectra thereby saving our resources.

There are already a huge amount of publicly available mass spectra of natural products and some articles about Deep learning on mass spectra data into which I want to dig deeper (mainly Tran N. H. et al., 2017 and 2019). It turned out to be possible to detect natural products by their mass spectra and also find new ones missing in the database using a high-throughput technology built on computational algorithms. I'm going to use this one hundred million tandem mass spectra in the Global Natural Products Social (GNPS) molecular networking infrastructure (Wang M. et al., 2016) to select peptide compounds and classify them using Machine learning algorithms. The labels can be taken from molecular structures from GNPS library (trustworthy labels manually obtained by biologists) or from highly-reliable DEREPLICATOR identifications. In both cases it's **several hundred cyclic and non-cyclic structures** and **several thousand spectra** related to them (3-5 different spectra for the structure on average).

Problem Statement

The problem of this Capstone project is to **categorize PNPs spectra** into spectra corresponding to **cyclic** and **linear** compounds (branch-cyclic and complex classes can also be considered). Thus the program requires spectrum of the unknown compound as input and defines type of the compound structure as output.

The workflow for approaching a solution given the problem includes

- **Collect** the data. Choose peptide not complex compounds from GNPS Public Spectral Library and also the same highly-reliable DEREPLICATOR identifications.
- **Preprocess** the data. Try different sizes of discretization step to vary number of features. We want to get computationally simpler model, but still with acceptable performance.
- **Split** the data into training, validation and test sets such that both linear and cyclic compounds fall into each of these sets in acceptable proportions.
- **Choose**, **train** and **tune** the model. At first make sure that such simple models as clustering do not work (try K-means, Gaussian mixtures or Hierarchical clustering). Secondly try SVC and at the end build first one CNN. Get some intuitions about

how the models work on spectra data by testing them and plotting some scores, varying layers and other hyperparameters, use different optimizers and etc.
- **Evaluate** the solution. Visualize some predictions. Compare trained models with each other by computing confusion matrix, Receiver operating characteristic and the area under this curve AUC.

Metrics

Confusion matrix is chosen to show how many and what type of mistakes the model makes on test dataset. *False* means that spectrum corresponds to other cyclicality than the model got. So *FP* is the number of spectra corresponding to non-linear compounds predicted as linear and *FN* means that actually linear compound is predicted as non-linear.

ROC curves and **AUC** will measure performance of the model instead of *accuracy* since the dataset can't be considered fully balanced and the model may have a large accuracy but be unfair owing to Accuracy paradox). In its turn AUC deals well with the small imbalance of input data.

II. Analysis

Data Exploration

Each spectrum is in the MGF Format consisting of list of pairs of mass-to-charge ratio and intensity (see `data/spectra/*.mgf`, `data/spectra_REG_RUN/*.mgf` or `data/GNPS-LIBRARY.mgf`). To make it clearer here is an example of such file:

```
BEGIN IONS
PEPMASS=712.31
CHARGE=0
MSLEVEL=2
SOURCE_INSTRUMENT=LC-ESI-qTof
IONMODE=Positive
NAME=Microcolin C M+Na
SMILES=CCCC[C@@H](C)C[C@@H](C)C(N(C)[C@@H](CC(C)C)C(N[C@H]([C@H](O)C)C(N(C)[C@H](C(C)C)C(N1[C@@H](C(N2C(C=C[C@@H]2C)=O)=O)
SPECTRUMID=CCMSLIB00000001660
SCANS=1
271.888367 17345.0
289.879761 28408.0
329.993896 100546.0
331.070801 33707.0
.
.
.
714.161499 1.0
END IONS
```

For **spectra from GNPS library** Marvin suite is used to get information about compound structure from the field of MGF file named SMILES.

```
molconvert mol:V3+H -s 'SMILES' -o Molfile
```

So the compound structure is in the Molfile containing information about the atoms, bonds and molecular coordinates (see `data/mols/*.mol`). It turned out that half of all spectra namely 2419 out of 4666 don't have SMILES and therefore were filtered out. Here is an example of Molfile:

```

  Mrv1920 10081913022D

  0  0  0     0  0            999 V3000
M  V30 BEGIN CTAB
M  V30 COUNTS 112 113 0 0 1
```

```
M  V30 BEGIN ATOM
M  V30 1 C 0.6359 -23.6727 0 0
M  V30 2 C 1.9696 -24.4427 0 0
M  V30 3 C 3.3032 -23.6727 0 0
M  V30 4 C 4.6369 -24.4427 0 0
M  V30 5 C 5.9706 -23.6727 0 0 CFG=1
M  V30 6 C 5.9706 -22.1327 0 0
.
.
.
M  V30 113 1 45 112
M  V30 END BOND
M  V30 END CTAB
M  END
```

Molfile helps to identify whether the spectrum corresponds to peptidic compound or not. Compound with number of components more than 3 is recognized as peptidic. Number of components is obtained using Natural Product Discovery tools by command line below. *443 spectra* correspond to the peptide compounds.

```
print_structure Molfile -C share/npdtools/ --print_rule_fragmented_graph

number of components : 6
0 C10H19O 155.144
1 C7H13NO 127.1
2 C4H7NO2 101.048
3 C6H11NO 113.084
4 C5H7NO 97.0528
5 C5H6NO 96.0449
number of bonds : 6
1 -NC> 0
2 -NC> 1
3 -NC> 2
4 -NC> 3
5 -NC> 4
5 -NC> 5
```

Finally, the following command is used to determine the type of the compound structure:

```
print_structure Molfile -C share/npdtools/ --print_structure

branch-cyclic
```

As a result, *85 linear*, *82 cyclic*, *71 branch-cyclic* and *205 complex* spectra were founded.

Information about spectra **structures identified by DEREPLICATOR** can be found in tab-separated values file `data/REG_RUN_GNPS/regrun_fdr0_complete.tsv`, where *LocalSpecIdx* field is the spectrum index in the file with *SpecFile* path. And this spectrum corresponds to a compound whose cyclicality is in the *Structure* field.

```
Dataset    Id  SpecFile    LocalSpecIdx    Scan    LocalPeptideIdx Name    Score   P-Value PeptideMass SpectrumMass    Ret
MSV000078556    0   REG_fdr0_spectra/MSV000078556.mgf   0   0   8568    L-Valyl-L-leucyl-L-prolyl-L-valyl-L-prol    9   1.
MSV000078556    1   REG_fdr0_spectra/MSV000078556.mgf   1   1   8568    L-Valyl-L-leucyl-L-prolyl-L-valyl-L-prol    9   1.
MSV000078556    2   REG_fdr0_spectra/MSV000078556.mgf   2   2   8568    L-Valyl-L-leucyl-L-prolyl-L-valyl-L-prol    9   1.
MSV000078556    3   REG_fdr0_spectra/MSV000078556.mgf   3   3   8363    a-Substance_Ib  8   3.4e-14 685.3910000000001   34
.
.
.
MSV000080116    9   REG_fdr0_spectra/MSV000080116.mgf   9   9   6225    Surfactin_1-Me_ester    15  1e-15   1049.7  1050.7
```

DEREPLICATOR identifies *7505 peptidic* spectra (*3101 linear*, *2681 cyclic*, *1692 branch-cyclic* and *31 complex*).

I decided to start with binary classification task. So I removed from consideration all spectra corresponding to complex and branch-cyclic compounds. And as it can be seen from above there is still small **imbalance of input data** (*3186 linear* and *2763 cyclic* spectra).

Exploratory Visualization

The plots below show **first 32 spectra and their discretizations** with various step sizes. Mass-to-charge ratio along the X axis ranges from 0 to 5 000 and is divided into 100, 500, 1 000, 5 000, 10 000 and 50 000 intervals. The intensities are along the Y axis and those of them that fall into the same discretization interval are summed up.

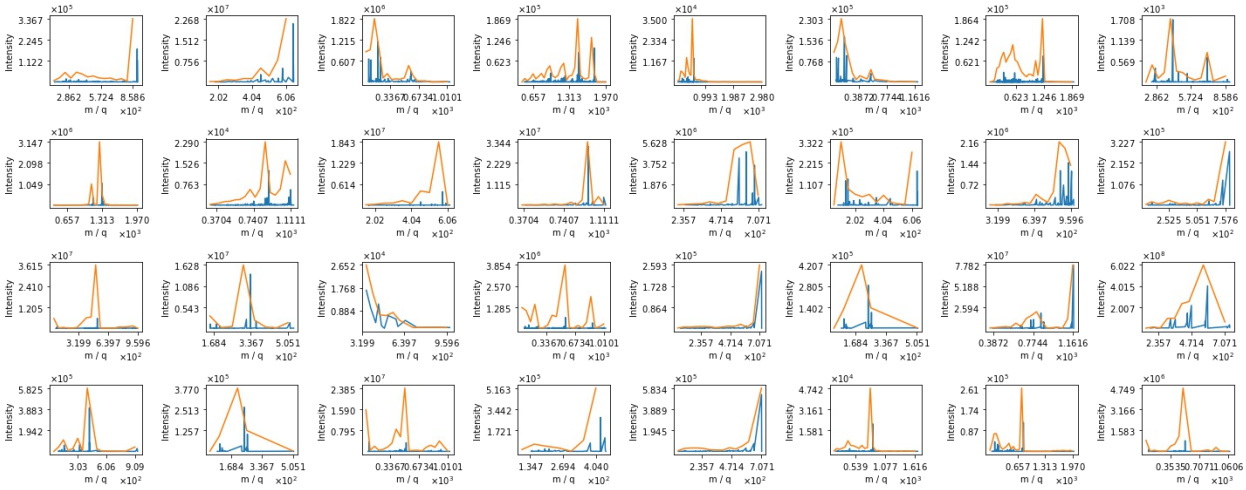

Fig. 1. Mass-to-charge ratio range is divided into **100 intervals**. The **dimensionality** of the problem is equal to **60** as intervals with zero intensity for all spectra from the dataset are not considered by the model as a features. Not all peaks are caught, only the general form of the spectrum can be guessed.

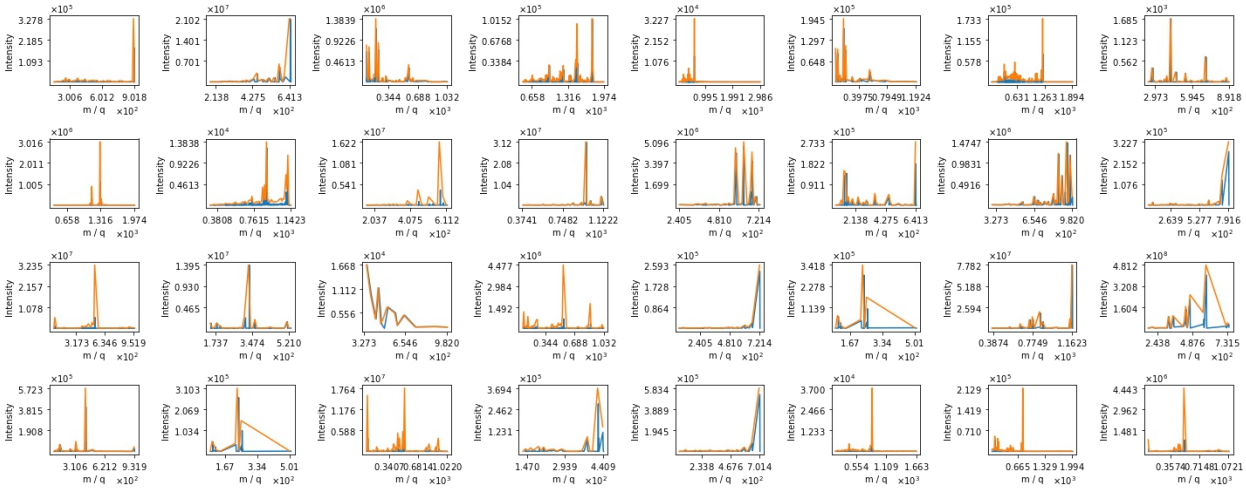

Fig. 2. Mass-to-charge ratio range is divided into **500 intervals**. The **dimensionality** of the problem is **297**. Most peaks of input spectrum are visible in discretization, but nearby peaks are still glued together.

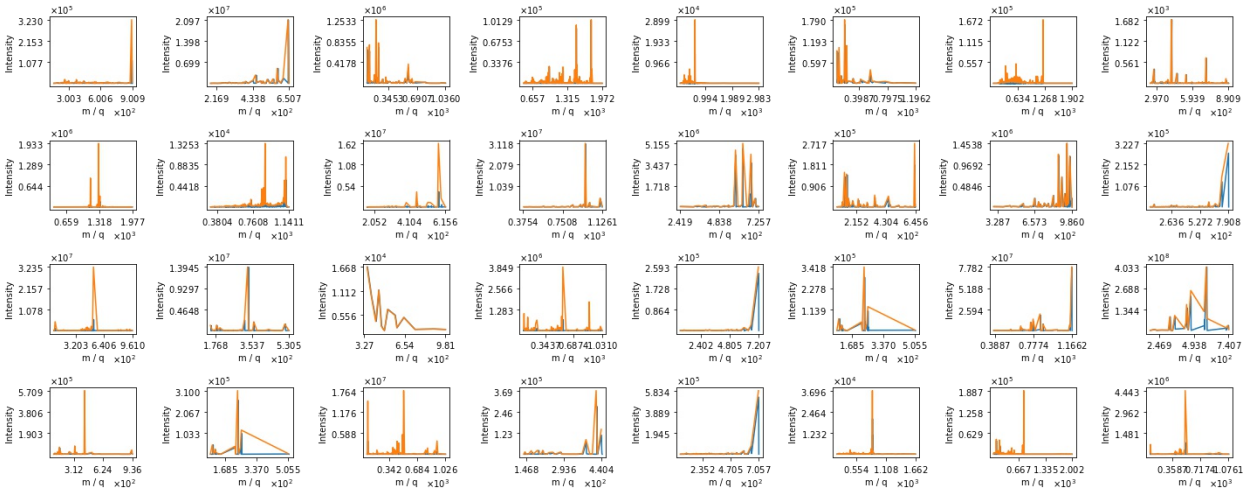

Fig. 3. Here is **1 000** intervals. The dimensionality is **594**. It can be seen that the discretization is still a bit closer to the original data and most likely the model will already be able to work well with such data.

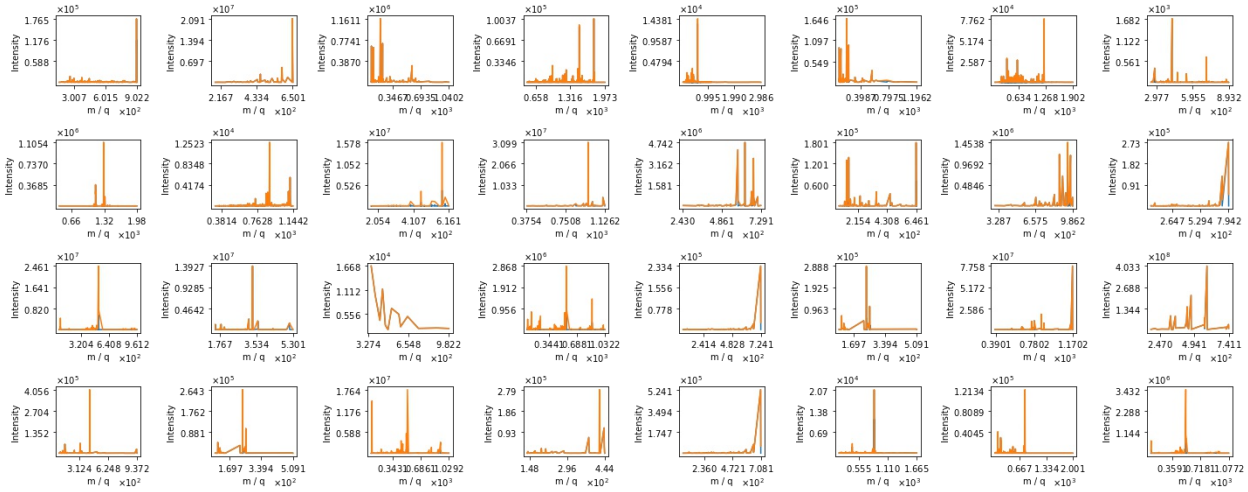

Fig. 4. Number of intervals is **5 000**, dimensionality is **2915**. In these plots the blue color (showing original spectra) is almost completely hidden from view - the discretization already fits original spectra data well.

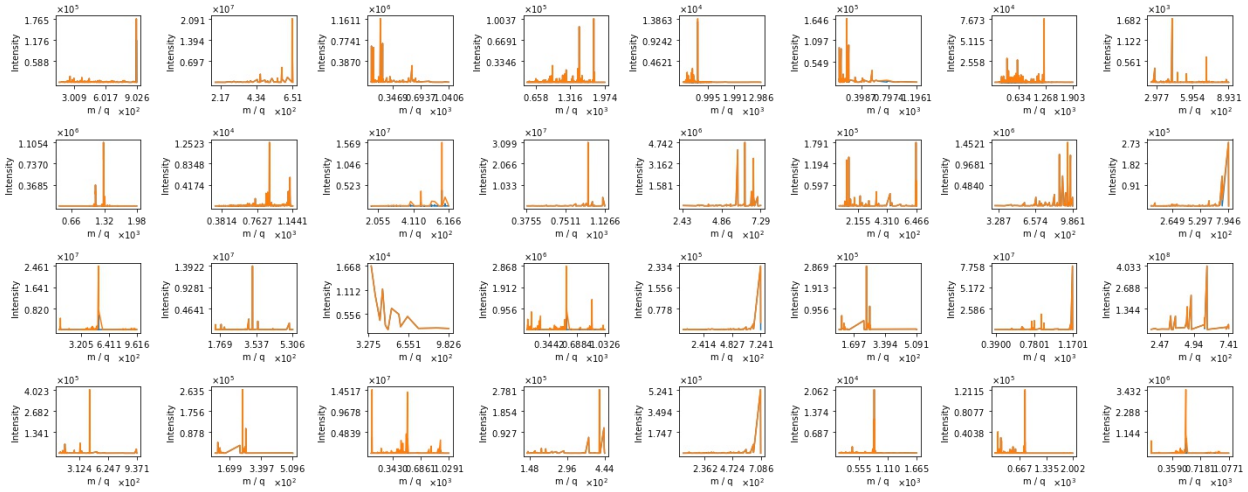

Fig. 5. Number of intervals is **10 000**, dimensionality is **5572**. Visually looks almost the same as the previous discretization. However, the dimension of the model is almost twice as big.

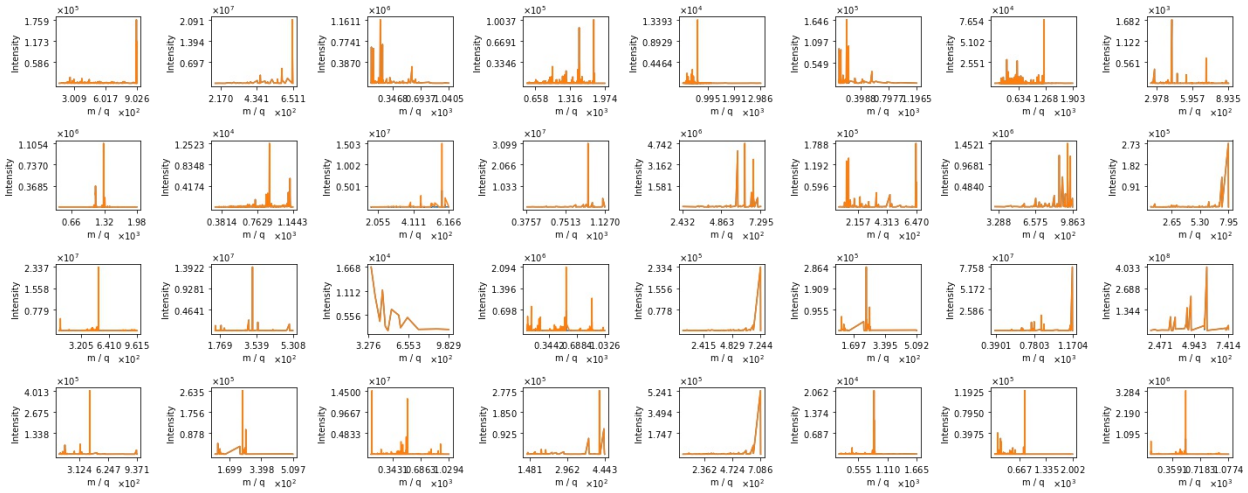

Fig. 6. Number of intervals is **50 000**, dimensionality is **23325**. Dimensionality is big, so let's stop here. Still not all peaks are caught - there are **783 055** different peaks in the original spectra.

We see that discretization with rather large step allows to recognize most of the peaks. This gives us an assumption that it's possible to **reduce the dimensionality** of the problem **not losing much** in quality at the same time.

Algorithms and Techniques

It's Supervised learning task because sample-label (namely spectrum-cyclicality) pairs exist. I will start with clustering algorithms (**K-means**, **Gaussian mixtures**, **Hierarchical clustering**) and **Support Vector Classification**. And then I'll try improve the result with **Neural Network**. There are two ways to work with such continuous spaces of input data: **discretize** the raw spectra or directly **approximate** them by functions. **CNN** should suite well for discretization as it allows to utilize spatial information. And **usual NN** will be used for function approximation. For now I will only focus on discretization. I plan to try various data representations and do some preprocessing steps (using different step sizes for discretization, summing up peaks within a single bin, replacing *NaNs* with zero in intensity vectors, removing zero features and etc).

- **Clustering** The number of clusters is known and equals 2 (linear and non-linear), other parameters will be left default.
- **Support Vector Machine** Default parameters will be used to have an overview of SVM performance on such discretized data. `class_weight` will be set to `balanced'` (to reflect small data imbalance).
- **Neural networks** The advantage of Neural networks approach is the possibility of non-linear models with respect to the features. CNN will include 2 convolutional layers (anyway up to 4 due to the large length of intensity vector), each with 64 filters of size 4 and two fully connected layers with 64 and 2 (number of output categories) neuron units, *tanh*/*ReLU* activation functions, max-pooling and dropout layers to prevent overfitting. I also will try a different models (various layers and etc.) and most **Keras** optimizers.

Benchmark

Used models can be measured by common metrics such as **AUC**, **precision**, **recall** and more since it is supervised learning problem. **Random model** will be used as a benchmark model for cyclic-linear classification.

III. Methodology

I will use **Python 3** with **pandas**, **NumPy**, **scikit-learn** and mainly **Keras**. All steps have already been done in `capstone.ipynb` - input data preprocessing, training, testing and model evaluation.

Data Preprocessing

Each spectrum will be converted into intensity vector by tiny step discretization in which mass-to-charge ratios are indices and intensities are values (let the length be less than 50 thousand). The function below creates binary file with **spectra dataframe** consisting of intensity vectors. Function input is **pathes to spectra files** in MGF Format, path to the output file and interval

bounds of mass-to-charges (on w hich it's necessary to split the values along the x axis).

```
def get_fthr(spectra_pathes, fthr, discrete_masses):
    spectra_df = pd.DataFrame()
    for mgf_file in tqdm(spectra_pathes):
        with open(mgf_file, 'r') as fin:
            fin.readline()
            header, intensity = get_spectrum(fin)
        id = os.path.splitext(os.path.basename(mgf_file))[0]
        bins = pd.cut(intensity[:, 0], bins=discrete_masses, labels=False)
        spectrum_df = pd.DataFrame({id: intensity[:, 1], 'binned': bins}).groupby(['binned']).sum().T
        spectra_df = pd.concat([spectra_df, spectrum_df], sort=True)
    spectra_df.columns = spectra_df.columns.astype(str)
    spectra_df.reset_index().to_feather(fthr)
    return fthr
```

NaN values are replaced w ith zero.

Implementation

All actual code is w ell formatted and presented in `capstone.ipynb`. But here are some additional details and most complicated functions.

The function below plots first 32 intensities vectors (function of the mass-to-charge ratio) and their discretizations:

```
def plot_discretisation(tst_d_m=discrete_masses):
    i = 0
    fig = plt.figure(figsize=(20, 8))
    for mgf_file in tqdm(spectra_pathes):
        if i == 32:
            break
        ax = fig.add_subplot(4, 8, i + 1, xticks=[], yticks=[])
        with open(mgf_file, 'r') as fin:
            fin.readline()
            header, intensity = get_spectrum(fin)
        id = os.path.splitext(os.path.basename(mgf_file))[0]
        ax.plot(intensity[:, 0], intensity[:, 1])
        bins = pd.cut(intensity[:, 0], bins=tst_d_m, labels=False)
        tst_df = pd.DataFrame({'intensity': intensity[:, 1], 'binned': bins}).groupby(['binned'], as_index=False).sum()
        tst_df['x'] = tst_df['binned'].transform(lambda b: tst_d_m[b])
        ax.plot(tst_df.x, tst_df.intensity)
        ax.set_xlabel('m / q')
        ax.set_ylabel('Intensity')
        ax.set_xticks(np.arange(max(tst_df.x) / 3, max(tst_df.x) * 7 / 6, max(tst_df.x) / 3))
        ax.set_yticks(np.arange(max(tst_df.intensity) / 3, max(tst_df.intensity) * 7 / 6, max(tst_df.intensity) / 3))
        ax.ticklabel_format(axis='both', style='sci', scilimits=(0, 0), useMathText=True)
        i += 1
    fig.tight_layout()
    return ax
```

The next code helps to visualize some predictions. Plot random 32 discrete spectra and w rite predicted and real types of the compounds structures (cyclic or linear) corresponding to these spectra.

```
# define text labels
names = ['linear', 'cyclic']

# plot a random sample of test images, their predicted labels, and ground truth
fig = plt.figure(figsize=(20, 8))
y_cnn = cnn_clf.predict(X_test_3d)
for i, rc_i in enumerate(np.random.choice(X_test.shape[0], size=32, replace=False)):
    ax = fig.add_subplot(4, 8, i + 1, xticks=[], yticks=[])
    x = np.array([discrete_masses[int(bin_i)] for bin_i in choosen_df.columns.values.tolist()])
```

```
    y = np.array(X_test[rc_i])
    y_mask = (y != 0)
    ax.plot(x[y_mask], y[y_mask])
    pred_idx = np.argmax(y_cnn[rc_i])
    true_idx = int(test_binary_labels[rc_i])
    ax.set_title("{} ({})".format(names[pred_idx], names[true_idx]),
                 color=("green" if pred_idx == true_idx else "red"))
```

Code cell below implements evaluation of compared models (Random model, SVC and CNN) on imbalanced data. Function `evaluate_imbalanced` get all needed characteristics for plotting *ROC* such as *False* and *True Positives* rates and also *AUC* score.

```
def evaluate_imbalanced(clf):
    type_clf = type(clf).__name__
    if type_clf == 'Sequential':
        x = X_test_3d
    else:
        x = X_test
    y_pred = clf.predict(x)
    if type_clf != 'SVC':
        pred_binary_labels = np.argmax(y_pred, axis=1)
        y_score = y_pred[:, 1]
    else:
        pred_binary_labels = y_pred
        y_score = y_pred
    tn, fp, fn, tp = confusion_matrix(test_binary_labels, pred_binary_labels).ravel()
    print('{name} TN = {tn}, {name} FP = {fp}, {name} FN = {fn}, {name} TP = {tp}'.
          format(tn=tn, fp=fp, fn=fn, tp=tp, name=type_clf))
    fpr, tpr, thresholds = roc_curve(test_binary_labels, y_score)
    auc_score = auc(fpr, tpr)
    return fpr, tpr, auc_score


fpr_d, tpr_d, auc_d = evaluate_imbalanced(dummy_clf)
fpr_cnn, tpr_cnn, auc_cnn = evaluate_imbalanced(cnn_clf)
fpr_svc, tpr_svc, auc_svc = evaluate_imbalanced(svc_clf)

# Plot ROC curves for these models
plt.figure()
plt.plot(fpr_d, tpr_d, lw=2, label='Dummy ROC curve (area = %0.2f)' % auc_d)
plt.plot(fpr_cnn, tpr_cnn, lw=2, label='CNN ROC curve (area = %0.2f)' % auc_cnn)
plt.plot(fpr_svc, tpr_svc, lw=2, label='SVM ROC curve (area = %0.2f)' % auc_svc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

CNN are accurate described in Results section therefore missing here but code implementation still can be founded in Notebook file both for CNN and for other models.

Refinement

- Removing *NaNs* is the first thing that definitely helped to improve model performance.
- Excluding complex and branch-cyclic classes greatly simplifies the task. 4-class classification model showed results comparable to the random model while binary classification model got *AUC* close to 1.
- Using only non zero features made the model faster. This allowed to run more complex models. So I switched to larger number of filters in CNN (from 4 to 64), then I tuned the kernel size.
- Using class weights in loss function helped to balance classes. Models with simple loss function mostly gave out the predominant class.
- Dropout layers helped to get comparable results on test and train sets - prevented overfitting.

IV. Results

Model Evaluation and Validation

I use **validation set** when training the model to tune parameters. And then evaluate the chosen model on **test unseen data**. The final architecture (how many and which layers for CNN) and hyperparameters were chosen because they outperformed all previously tried models.

The final model

- Takes an input vector of **length 5572**. This is achieved by discretization into 10 000 intervals.
- Consists of 2 **Convolutional** layers. Both learn 64 filters. The length of the filters of these layers is 4. The activation function is *ReLU*.
- After each of the Convolutional layers is **Max pooling** operation with pooling window size 2.
- The first **Fully connected** layer has 64 outputs and *RelU* activation function, the second - 2 and *Softmax*. It corresponds to the two output classes, linear and cyclic.
- Fraction of the input units to drop equals 0.3 after the last Max pooling layer and 0.4 between two Fully connected layers.
- Compile with `categorical_crossentropy` loss function and `rmsprop` optimizer.
- Fitting using **4759 points** (train on 3807 samples, validate on 952 samples) with number of samples per gradient update equal 32 on **25 epochs** (shuffle the training data before each epoch).
- Use **balanced class weights**.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_15 (Conv1D)           (None, 5572, 64)          320
max_pooling1d_15 (MaxPooling (None, 2786, 64)          0
conv1d_16 (Conv1D)           (None, 2786, 64)          16448
max_pooling1d_16 (MaxPooling (None, 1393, 64)          0
dropout_15 (Dropout)         (None, 1393, 64)          0
flatten_8 (Flatten)          (None, 89152)             0
dense_15 (Dense)             (None, 64)                5705792
dropout_16 (Dropout)         (None, 64)                0
dense_16 (Dense)             (None, 2)                 130
=================================================================
Total params: 5,722,690
Trainable params: 5,722,690
Non-trainable params: 0
```

To verify the **robustness** of the final model I run the process more than 20 times on different sets including random shuffle, input space, number of spectra and the results changed very slightly.

The model have exceeded all my expectations. The results are much better than the results obtained by the random model and, moreover, are close to 1. **AUC** is equal to **0.97** (True negative = 614, **False positive = 24**, **False negative = 9**, True positive = 543).

Justification

Final CNN solution gets 33 errors on 1190 spectra test set comparing with benchmark model that gets 552 errors. SVC shows results close to CNN however lying more times (40 errors) on this test set. So the final results found are **stronger than the benchmark**, outperform other models and significant enough to **solve the problem** since AUC shown on the plot below is very close to ideal.

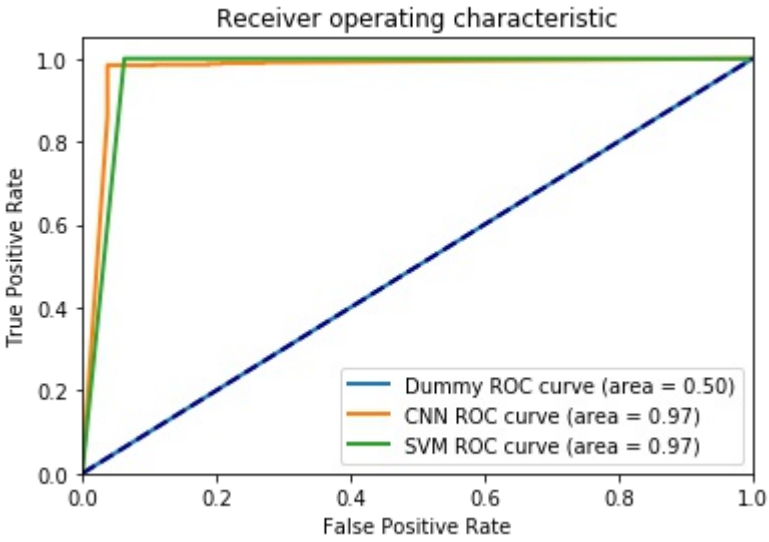

Fig. 7. ROC curves and **AUC** demonstrate the performance of three models: **dummy** classifier (Benchmark random model), chosen **CNN** and **SVC** (clustering classification didn't give any significant results). CNN gets the best results but is comparable with SVC.

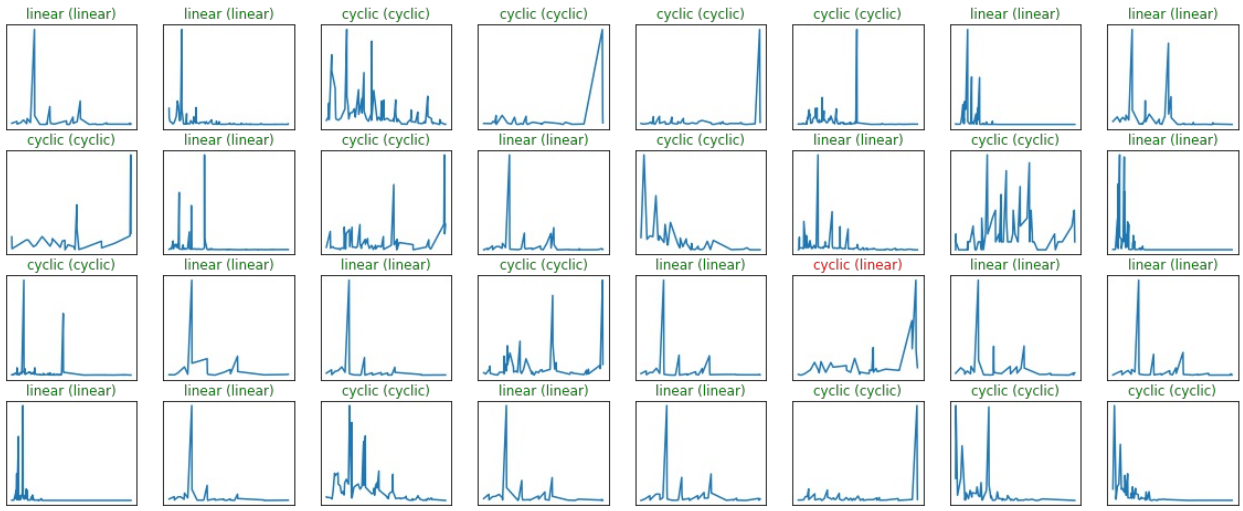

Fig. 8. Discretized spectra, **predicted** type of the compound structure (cyclic or linear) corresponding to them and the **true** type in brackets. Green labels mean true prediction, red where the model made a mistake.

V. Conclusion

Free-Form Visualization

The size of intensity vector affects the results of all models. Except random of course, it has *AUC* equal to 0.5. Increasing the number of recognized peaks in input spectra highly improves SVC and slightly improves CNN too. For small discretization steps (10 000 and 50 000 intervals) CNN and SVC *AUC* match and equal 0.97 and 0.98 respectively. For large discretization steps (from 100 to 5000 intervals) CNN outperforms SVC (*AUC* values are 0.96 vs 0.81, 0.97 vs 0.81, 0.96 vs 0.92 and 0.98 vs 0.96). So performances of both models are on a par with each other. CNN gets better results but on more thorough discretization SVC catches up with CNN.

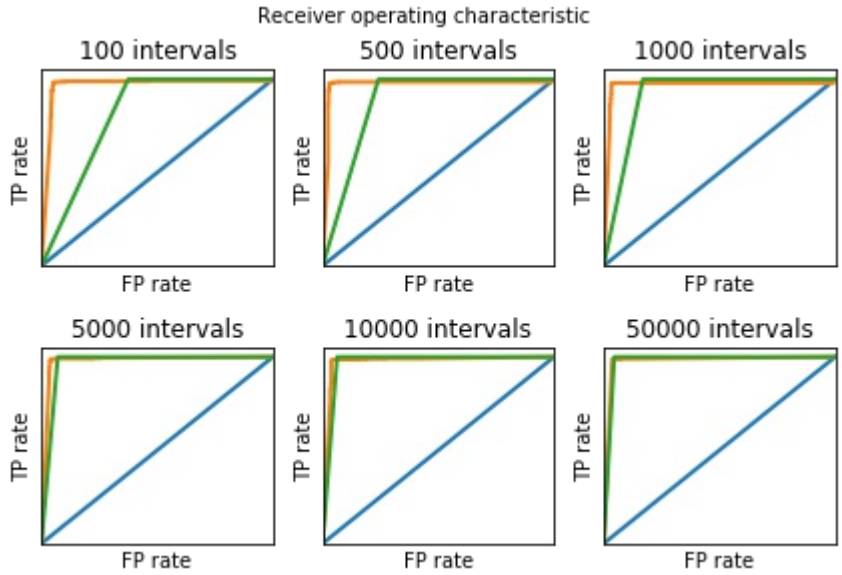

Fig. 9. ROC curves for different step sizes of input spectra discretization.

Reflection

Initially I didn't have any understanding of whether Machine learning can be used to predict type of unknown compound structure by its spectrum. I didn't know if it is difficult task or not. I didn't know how many intervals I need to use and how many layers, filters and etc. It's very interesting to get such good scores for this problem. I want to try to solve many different tasks using Machine learning and also I think that it's possible.

Improvement

Next task I want to solve is to improve DEREPLICATOR results. Benchmark model is **current DEREPLICATOR**. After classifying spectra by obtained CNN I plan to run DEREPLICATOR for cyclic spectra only against cyclic compounds and separately for linear spectra only against linear compounds. And then compare FP and elapsed time for these results and for DEREPLICATOR on full set of spectra. A good result that relates to the domain of Natural products identification would be less elapsed time and less FP at the same time obtained by **target matching DEREPLICATOR** (cyclic spectra against cyclic compounds and linear against linear) than by current DEREPLICATOR pipeline. It will also confirm that the model correctly classify the spectra by their structures into two groups.

AUC, **precision**, **recall**, **F1 score** and **FP** are a good choice of evaluation metrics that can be used to quantify performance of both the current DEREPLICATOR (in the sense of benchmark model) and the target matching DEREPLICATOR. Here FP means that DEREPLICATOR got a structure that actually doesn't match input spectrum.