**An open-source machine learning pipeline tuned for predicting peptide therapeutic activities**

**Written by Peter Sarvari, Visiting Undergraduate Researcher at Research Laboratory of Electronics, Massachusetts Institute of Technology, MEng student at Imperial College, London**

**Supervised by Dr Cesar de la Fuente Nunez and Dr Raul Castro Fernandez**

***Abstract****.* *We identified a machine learning procedure, which allows for very accurate, easily-interpretable and realistic prediction of the potential therapeutic effects of small sequence peptides. A Machine Learning pipeline was developed in Python to predict peptide activity consisting of the following 5 steps: (1) down sampling of the negative (no therapeutic activity) dataset to achieve class balance, (2) removing >90% similar sequences from the datasets, (3) extraction of the baseline (amino acid composition, 20 features), 82, 114 or 281 features, (4) training state of the art neural network model in the keras framework and (5) evaluation of model performance using one of the 7 options that can be chosen by the user and 5 metrics to report performance. Using this pipeline, we got above 80% accuracy in the prediction of all of the 7 therapeutic activity we tested, namely (1) antiviral, (2) cell penetrating, (3) antibacterial, (4) anticancer, (5) toxic, (6) tumour homing and (7) antihypertensive. For (1) we found no previous prediction in the literature and we achieved 0.61 Matthew’s Correlation Coefficient (MCC) and 80.5% accuracy. We exceeded the to-date highest known prediction accuracy for cell penetrating, tumour homing, antibacterial, toxic and antihypertensive peptides, obtaining 0.77 MCC with 89.9% accuracy, 0.72 MCC with 85.8% accuracy, 0.76 MCC with 88.0% accuracy, 0.72 MCC with 86.7% accuracy and 0.78 MCC and 89.3% accuracy, respectively using this developed rigorous protocol. We further came up with a method to deduce some of the features whose high values help obtaining the desired protein function. We named this method ADENOLIM. The user manual describing how to use our pipeline on your PC is included in the supplementary information. It details the installation of Python (Conda) on Windows as well as all the relevant libraries and the setting up of the pipeline on a remote computer via Google Cloud. The link to the GitHub page of the project is: “https://github.com/sarvarip/Protein-prediction-pipeline”.*

***Introduction****.* Peptide-based drug discovery is an option to tackle new therapeutic challenges. [1] The idea to use peptides as a defence against microbials comes from nature: “host defence antimicrobial peptides are key components of the innate immune system shared by both invertebrates and vertebrates.” [2] Here we investigate 7 of the many distinct functions peptides have and these 7 are particularly useful in peptide-based drug design.

* Antibacterial peptides provide a promising alternative to antibiotics, which have serious side effects including killing microbiota and are becoming less and less efficient due to emerging antibiotic-resistant infections. [3], [4]
* Cell-penetrating peptides are useful tools to get materials inside the cell [5], for example, Phosphorodiamidate morpholino-oligomers (PMOs), used in antisense strategies, can be delivered into cells via conjugation with cell-penetrating peptides. [3]
* Tumour homing peptides (THPs), home specifically to tumour specific microenvironment [6], are used to deliver therapeutics to the tumour site.
* Anticancer peptides do not affect the normal physiological functions of the body like the current, expensive cancer treatment does and there is no resistance known that can be built against them. [7] Hence, they could be alternatives to the currently available cancer treatment.
* Antiviral peptides can reduce the binding of viruses to host cells as well as make viruses unable to infect host cells by various mechanisms. [8] These properties enable them to be used as drugs against viral infections.
* One bottleneck in peptide-based drug design is the potential toxicity of the peptide. [9] Hence a software giving clues whether a certain proposed sequence has toxic properties could save a lot of time in drug design.
* Peptides have also been shown to be effective in reducing blood pressure, hence can be used as medications for high blood pressure, which threatens millions of lives. [10]

There have been many attempts in the literature to predict therapeutic activities of peptides and many of these predictions have been successful. However, we have seen no consensus on the features extracted and the methods used, mainly because it is widely believed that different feature extraction methods will work the best in different cases. This suggests that building a machine learning model in proteomics is slow and requires a lot of effort much of which is burnt on the so-called “feature engineering”. If features were found, which generally describe small peptide sequences well enough to give satisfactory results after training for multiple purposes, it would make statistical analysis of peptides much more efficient. Consensus on the methods is equally important, because otherwise the results are difficult to compare, unrealistic and easy to misinterpret.   
Amino-acid composition [4]–[6], [11], dipeptide composition [5], [9], dipeptide gaps [7], physiochemical properties [5], [12], binary profile of patterns [5], z-descriptors [13] and pseudo amino acid composition [7], [14], [15] are examples of the features that many groups have used to build the machine learning models. One reason many different methods exist in the literature for feature extraction is that the features that produce the best result in one activity case may not be adequate for a good prediction in another case. In other words, no general features have been found that can characterize the sequences well enough to discern multiple therapeutic activities. Here we propose a protocol and its realization in Python programming language, which can provide good prediction for all the 7 different therapeutic activities tested using the same features inputted into the same machine learning model. This is not only giving us clues about important and general peptide descriptors and following up on the QSAR project [16], but also offers an easy-to-handle customizable proteomics tool that can quickly build state-of-art prediction model given a positive and negative sequences dataset. Using the python framework makes it very easy to share the code with many people in the community, use the code and enjoy all its benefits and add changes to further improve the pipeline. Ultimately, we aim to make this an open-source project, create a user interface and maintain a web-server that would take the user inputs, run the Python script and return the outputs.

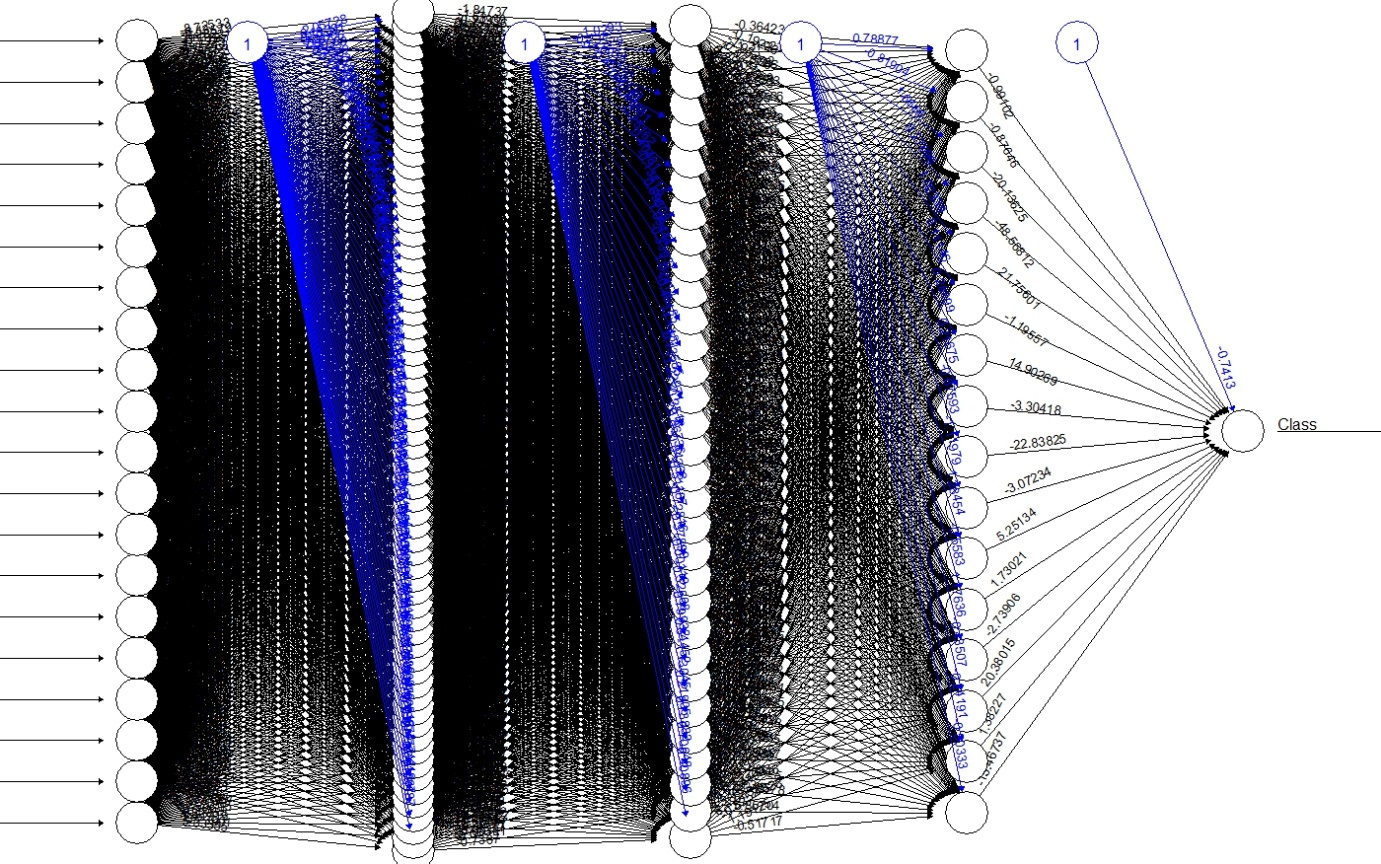
***Methods****.* The machine learning pipeline consists of the following 5 main stages: (1) down sampling of the negative dataset to achieve class balance, (2) removing >90% similar sequences from the datasets, (3) extraction of the baseline (20), 82 or 114 features, (4) training state of the art neural network model in the keras framework and (5) evaluation of model performance using one of the 7 options that can be chosen by the user.  
Many peptide activity predictions in the literature do not remove similar sequences from the positive dataset. This can lead to overestimation of the performance of the machine learning model, since it is much easier to decide which class a random sample belongs to if there is a very similar sequence already in one of the classes. The importance of removal of more than 90% similar sequences is also discussed in Chou et al. [7] Note that the negative dataset may also contain similar sequences, even if it consists of randomly chosen sequences from big protein databases, for example, UniProt. We have integrated a similar sequence (>90% identity) removal in the pipeline. If the user is confident that the datasets do not contain similar sequences, then this step can be skipped, but otherwise we recommend similar sequence removal to ensure that the model performance is realistic. There is one exception: if the negative dataset is specific (consists of used-to-be positive peptides slightly changed such that they lose their therapeutic activity) and hence contains similar sequences to the ones in the positive dataset as well, then there is no need to remove them, in fact, removing them would be detrimental, since we would be removing the key samples that reflect on subtle but important differences that are crucial for the therapeutic activity.  
Regarding the Machine Learning methods, Support Vector Machine (SVM) is well-represented in the literature [10], [4]–[7], [9], [12], [13], [17]. SVM being a Kernel machine results in costly training and poor generalization in some cases, especially if the number of training samples is small. With deep learning approaches the number of defined regions can grow exponentially with the number of training examples. [18] Also, Hinton et al. demonstrated that deep learning approaches outperform SVM using the MNIST benchmark dataset. [16] Hence in the developed protocol, we use neural networks as the machine learning algorithm. In addition, we included two recently published strategies, dropout [19] and batch normalization [20] to avoid overfitting and to accelerate the learning process. We use the Adam optimization algorithm [21] to find the weights of the model associated with the smallest cost. This is the recommended optimization by Andrej Karpathy [22]. The momentum parameter was set to 0.95 and the learning rate was set to 10 times the default value in the keras module. These values are recommended by the dropout paper. The dropout rate in the hidden layers was chosen to be 0.3. We observed that the rate does not have significant effect on the accuracy of the model if it is reasonable (0.2-0.6). The dropout rate in the input layer was set to 0.2. The rectified linear unit (ReLU) [22] activation function is used to introduce nonlinearity between the layers and the network has 3 hidden layers with 64, 32 and 16 neurons, respectively (see Figure 1, graph produced by neuralnet package in R [23]).   
Evaluation of the trained machine learning model is another step in the pipeline that needs a consensus, without which the results can be misleading and cause ambiguity. For example, a common pitfall is to tune hyperparameters using the test set and then report the best result as the performance of the model, rather than assessing the performance of the model using the chosen hyperparameter on a previously unseen (independent) set [24]. Even without hyperparameter tuning, lots of confusion can be created by reporting the accuracy on the training set as the best result. Training accuracy has not much to do with the usefulness of the model: a completely overfitted model would have 100% training accuracy, but it should be avoided [25], since the model would be useless and would give bad predictions on previously unseen samples. But even if these traps are avoided, many methods exist for estimating the performance of the model on independent dataset. [25] In the literature, the leave-one-out cross-validation (jackknife) procedure is used widely. [7] However, there are two major drawbacks of this procedure: first, it requires a lot of computational effort and second, it is likely to have high variance. [25] It is a special form of K-fold cross validation, where K=N, where N is the number of samples in the dataset. Unless N is small (e.g. N<100-200) K=5 or K=10 is recommended. [25] In the developed protocol, we provide 7 different well-documented options for approximating the model performance. The default (1) is the recommended K-folds with K=10. If the dataset is big (N>1000-2000) one might start wondering if the simple holdout cross validation (2) is adequate. One can also manually refer to an independent test set (3) in the script. Other options include partitioning the dataset into 3 to do hyperparameter tuning (4) with hold-out cross validation and merging one dataset and an independent dataset to do K-folds cross-validation (5) or hyperparameter tuning (6). Additionally, with option (7), old saved model can be loaded to predict the probabilities and classes of new samples.   
It is not just the evaluation method that can cause confusion, but also the metric. Bad examples include using accuracy as a metric, whilst doing binary classification on an imbalanced dataset. We handle this problem in two ways. First, we always work with balanced (or almost balanced) dataset and we integrated down sampling of the negative dataset into the pipeline, such that the classes become balanced. This is not as easy as it may seem, because we want the samples in the classes to be equal after the similar sequences have been deleted and the junk samples have been discarded. However, down sampling after the similar sequence removal would mean a lot of extra processing work for nothing. We solved this issue by down sampling multiple times and by making reasonable assumptions. Working with balanced dataset is also beneficial for the learning [26]. Second, apart from accuracy, we also report F1 score, precision, recall and Matthew’s correlation coefficient (MCC). These measures make sure that every prediction made using the pipeline is clear, easy-to-interpret and directly comparable to other predictions.   
Feature extraction takes significant amount of time when features other than the amino acid composition are extracted. To make the pipeline more time-efficient, we parallelized the feature extraction in a manner that each process computes features for one sample at a time. The extracted features are stored in a proxy object that is pickable, so the array of descriptors can be saved on the disk and used later. The speed gain is maximally the number of cores, but is expected to be less because of the computational overhead of copying the descriptors from the different processes to the proxy object.

Figure 1 - Baseline Neural Network with 20 inputs (amino acid composition), 3 layers with 64, 32 and 16 neurons, respectively

The features. The default 82 features are extracted using the pydpi library [27] and the script featurex\_v3. The features include

* Pseudo Amino Acid composition [15] (with lambda = 3; weight = 0.05, taking residue mass, hydrophobicity, hydrophilicity, pK1, pK2 and pI into account; 20 features describing general amino acid composition and 3 added extra features)
* kappa shape descriptors (7)
* charge descriptors (25)
* molecular property descriptors (6) and
* topological information indices proposed by Basak (21).

We also experimented by adding the following 32 features to the default, yielding 114 features.

* Moreau-Broto autocorrelation (32)

We further increased the number of features by 167 in the script featurex\_v3\_pseaac\_aac by adding:

* CDT (composition, transition and distribution) [32] (147)
* Amino acid composition (20)
* Our own descriptors describing the average and variance of 5 amino acid properties: hydrophobicity, hydrophilicity, pK1, pK2, pI (10)
* Moreover, we deleted the following 10 features from the charge descriptor: maximum positive and negative charge on all, oxygen, hydrogen, nitrogen and carbon atoms (-10)

Data-mining. A lot of effort has been made to collect as many samples for each of the 6 datasets as we could find online. This is to assure that the results are a good estimate of the real performance of the pipeline.

* Antiviral – Number of samples after removing similar entries: 2142
  + Positive: taken from
    - SATPdb: <http://crdd.osdd.net/raghava/satpdb/down.php> [28] and
    - UniProt: [www.uniprot.org](http://www.uniprot.org), looking for reviewed sequences of length between 4 and 42 with a keyword of antiviral protein
  + Negative: taken from
    - UniProt: [www.uniprot.org](http://www.uniprot.org), looking for reviewed sequences of length between 4 and 42 without a keyword of antiviral protein
* Cell penetrating – Number of samples after removing similar entries: 633
  + Positive: taken from
    - cellppd: <http://crdd.osdd.net/raghava/cellppd/dataset.php> by merging the sequences found in CPPsite1, CPPsite2 [29], Independent, Sanders2011-b [12], Dobchev-2010 [30], Hansen-2008 [13], Hallbrink-2005 [17]
  + Negative: taken from
    - UniProt: [www.uniprot.org](http://www.uniprot.org), looking for reviewed sequences of length between 4 and 34
* Antibacterial – Number of samples after removing similar entries: 2896
  + Positive: taken from
    - SATPdb: <http://crdd.osdd.net/raghava/satpdb/down.php>
    - UniProt: [www.uniprot.org](http://www.uniprot.org), looking for reviewed sequences of length between 4 and 100 and annotated as having antibacterial function
    - APD database [28] latest version with discretion of Dr Wang and his team
  + Negative: taken from
    - UniProt: [www.uniprot.org](http://www.uniprot.org), looking for reviewed sequences of length between 4 and 100 and not annotated as having antibacterial function
* Anticancer – Number of samples after removing similar entries: 752
  + Positive: taken from
    - SATPdb: <http://crdd.osdd.net/raghava/satpdb/down.php> ,
    - cancerPPD: <http://crdd.osdd.net/raghava/cancerppd/downseq.php> [11] (under sequences having natural amino acids),
    - sequences from the supplementary information of Chen et al. [7],
    - anticp: <http://crdd.osdd.net/raghava/anticp/datasets.php> [31] by merging the sequences found in main and Independent:1 dataset from and
    - UniProt: [www.uniprot.org](http://www.uniprot.org), looking for reviewed sequences of length between 4 and 50 and annotated as having anticancer function
  + Negative: taken from
    - UniProt: [www.uniprot.org](http://www.uniprot.org), looking for reviewed sequences of length between 4 and 50 and not annotated as having anticancer function
* Toxic – Number of samples after removing similar entries: 2966
  + Positive: merged from
    - SATPdb: <http://crdd.osdd.net/raghava/satpdb/down.php> and
    - UniProt: [www.uniprot.org](http://www.uniprot.org), looking for reviewed sequences of length between 4 and 50 and annotated as having toxic dose or as venom.
  + Negative: taken from
    - UniProt: [www.uniprot.org](http://www.uniprot.org), looking for reviewed sequences of length between 4 and 50 and not annotated as having toxic dose or as venom
* Tumour homing – Number of samples after removing similar entries: 626
  + Positive: taken from
    - TumorHPD [6] by merging the main and small datasets: <http://crdd.osdd.net/raghava/tumorhpd/down.php>
  + Negative: taken from
    - UniProt: [www.uniprot.org](http://www.uniprot.org), looking for reviewed sequences of length between 4 and 24
* Antihypertensive – Number of samples after removing similar entries: 924
  + Positive: taken from
    - SATPdb: <http://crdd.osdd.net/raghava/satpdb/down.php>
  + Negative: taken from
    - UniProt: [www.uniprot.org](http://www.uniprot.org), looking for sequences of length between 4 and 20

***Results and discussion.***

Here we publish the best results achieved by any of the three feature extraction method. However, using only one of feature extraction methods would result only in a 1-2% lower accuracy and only in some cases: see additional results in the Supplementary Information (Part 5 and 6).   
Results using the 82, 114 or 281 features and 10-folds cross-validation (option 0):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Prediction according to ML protocol | Matthew’s Correlation Coefficient (MCC) | Accuracy (%) | F1 score | Precision | Recall |
| Antiviral3 | 0.61(+/-0.04) | 80.45(+/-2.13) | 0.81(+/-0.04) | 0.81(+/-0.03) | 0.81(+/-0.02) |
| Cell penetrating1 | 0.77(+/-0.04) | 89.89(+/-1.48) | 0.87(+/-0.04) | 0.88(+/-0.02) | 0.89(+/-0.02) |
| Antibacterial3 | 0.76(+/-0.02) | 88.00(+/-1.37) | 0.87(+/-0.01) | 0.88(+/-0.01) | 0.88(+/-0.01) |
| Anticancer3 | 0.63(+/-0.05) | 81.56(+/-3.89) | 0.81(+/-0.05) | 0.81(+/-0.03) | 0.81(+/-0.03) |
| Toxic1 | 0.72(+/-0.02) | 86.72(+/-1.55) | 0.85(+/-0.02) | 0.86(+/-0.01) | 0.86(+/-0.01) |
| Tumour homing2 | 0.72(+/-0.06) | 85.83(+/-4.89) | 0.87(+/-0.04) | 0.86(+/-0.03) | 0.86(+/-0.03) |
| Antihypertensive3 | 0.78(+/-0.05) | 89.27(+/-2.70) | 0.89(+/-0.04) | 0.89(+/-0.03) | 0.89(+/-0.03) |

1: using the 82 features.   
2: using the 114 features.  
3: using the 281 features.

Results using only amino-acid composition (20 features) and 10-fold cross-validation (option 0):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Prediction according to ML protocol | Matthew’s Correlation Coefficient (MCC) | Accuracy (%) | F1 score | Precision | Recall |
| Antiviral | 0.51(+/-0.04) | 78.18(+/-2.91) | 0.70(+/-0.03) | 0.74(+/-0.02) | 0.75(+/-0.02) |
| Cell penetrating | 0.76(+/-0.03) | 89.20(+/-3.02) | 0.86(+/-0.02) | 0.88(+/-0.02) | 0.88(+/-0.02) |
| Antibacterial | 0.72(+/-0.04) | 87.59(+/-3.09) | 0.84(+/-0.03) | 0.85(+/-0.02) | 0.86(+/-0.02) |
| Anticancer | 0.56(+/-0.05) | 79.84(+/-3.26) | 0.75(+/-0.05) | 0.77(+/-0.03) | 0.78(+/-0.02) |
| Toxic | 0.68(+/-0.02) | 83.52(+/-1.92) | 0.84(+/-0.02) | 0.84(+/-0.01) | 0.84(+/-0.01) |
| Tumour homing | 0.63(+/-0.06) | 82.61(+/-4.89) | 0.80(+/-0.06) | 0.81(+/-0.03) | 0.81(+/-0.03) |
| Antihypertensive | 0.66(+/-0.05) | 87.31(+/-3.04) | 0.77(+/-0.05) | 0.81(+/-0.03) | 0.83(+/-0.03) |

The results show that prediction with the 82, 114 or 281 features is better than the prediction with only the 20 amino-acid composition descriptors in all cases. However, in the antibacterial and cell penetrating cases the difference is negligible. The results above suggest that it is advisable to use the amino-acid composition (AAC) as a baseline prediction, because AAC extraction is fast and in certain cases the MCC and accuracy achieved using AAC is very close to the MCC and accuracy obtained by using a lot more features.

In all the 7 examined cases, our best result is above 0.61 MMC and 80% accuracy, which indicates that the developed rigorous machine learning procedure gives satisfying results. For antiviral peptides, we found no previous prediction in the literature and we achieved 0.61 Matthew’s Correlation Coefficient (MCC) and 80.5% accuracy. We exceeded the to-date highest known prediction accuracy (on independent datasets) for cell penetrating, tumour homing, antibacterial, toxic and antihypertensive peptides, obtaining 0.77 MCC with 89.9% accuracy, 0.72 MCC with 85.8% accuracy, 0.76 MCC with 88.0% accuracy, 0.72 MCC with 86.7% accuracy and 0.78 MCC and 89.3% accuracy, respectively. This is summarized in the following table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Case | MCC | Accuracy (%) | Ref. MCC | Ref. Accuracy (%) | Ref |
| Cell penetrating | 0.77(+/-0.04) | 89.89(+/-1.48) | 0.63 | 81.31 | [5] |
| Tumour homing | 0.72(+/-0.06) | 85.83(+/-4.89) | 0.68 | 83.73 | [6] |
| Antibacterial | 0.76(+/-0.02) | 88.00(+/-1.37) | Unknown | 87.55 | [4] |
| Toxicity | 0.72(+/-0.02) | 86.72(+/-1.55) | 0.52 and 0.67 | 75.79 and 87.87 | [9] |
| Antihypertensive | 0.78(+/-0.05) | 89.27(+/-2.70) | 0.68 | 84.21 | [10] |

For the anticancer prediction, better results have been achieved using pseudo amino acid composition with G-gap dipeptide mode, see [7].

Building a protein therapeutic function classifier is undoubtedly useful and has the potential to have a substantial impact on future pharmaceutical design. Since our methods achieved high accuracy and MCC, medical companies can use our software to significantly reduce the amount of lab experiments involved in their design. Hence our software opens the door to a faster production line and eventually more effective medicines in shorter time. In addition to the software, here, we offer an insight into the features that are causing the sequences to have the function they have. This can guide sequence design even before using the software and hence can reduce the number of cases, where the software predicts that the desired function is not achieved and further speed up production of new protein-based medicines. We estimated feature function characteristic using our own method, here referred to as average derivative estimation in non-linear models (ADENOLIM). This method uses the idea that in linear regression models, feature importance can be very easily seen by examining the coefficient that multiplies the given feature. That coefficient is the partial derivative of the model with respect to the given feature. In case of neural network – since the model is much more convoluted – the derivatives with respect to (w.r.t) a given feature are not constants. Also, taking the derivatives is not trivial, and the backpropagation algorithm needs to be used for efficient computation [33]. Nevertheless, we can calculate the derivative of the output of the neural network w.r.t each of the features using the network built for each sample and then by summing the slopes through all samples, we get a number for each feature (ADENOLIM number), which represents the average slope w.r.t it. If we had infinite samples, then after the summation we would yield relatively small numbers for features, which are unimportant (meaning that they do not affect the output from the network much) or which are capricious (meaning that they do affect the output but sometimes positively, sometimes negatively). These features are not useful for us. However, features with large positive or large negative numbers after the summation are likely to most commonly contribute the sequence to have or not to have the therapeutic function, respectively. We can even investigate whether the slopes w.r.t a feature have the same sign in all cases or not (see Supplementary Information). Now, by increasing the value for a feature whose ADENOLIM number is big, we are more likely to yield a sequence with the desired therapeutic function. Obviously, even if in all cases the derivative w.r.t a feature is positive, it is no guarantee that the derivative will be positive in every possible case and we do not intend to claim such thing. We simply say that in such case, if the number of samples is significant (at least a 4-digit number) and they cover a fair amount of the feature space, then a feature with a big ADENOLIM number is more likely to yield a positive derivative w.r.t it in a new case. In general, the simpler the model is and the more the engineered new samples resemble to the old ones, the better this method works (it works perfectly for linear regression, which is the simplest model). In the following, we demonstrate how we can gain insight into the characteristics that some features bring to the samples in our protein function prediction pipeline.   
In the antibiotic case, using the amino acid composition features and the ADENOLIM method, we gained the following table, where the feature is the proportion of the amino acid represented by the one-letter code and the value is ADENOLIM number of the feature.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Feature | Value | Feature | Value | Feature | Value | Feature | Value |
| M | -3125 | V | -1056 | A | -269 | W | 800 |
| D | -2792 | Q | -1006 | I | 21 | G | 1001 |
| E | -2435 | Y | -840 | N | 120 | F | 1271 |
| T | -1766 | L | -515 | R | 124 | C | 2002 |
| P | -1114 | S | -313 | H | 216 | K | 3649 |

To build sample sequences that lack the antibiotic function, we chose amino acids M, D, E, T, P, V and Q, because they have significant negative values (cut off arbitrarily chosen to be – 1000). To build sequences with antibiotic properties, we chose amino acids G, F, C and K, which have the 4 highest ADENOLIM value. We randomly came up with 15 negative (no antibiotic function) and 15 positive sequences and used option 6 in the software we developed to predict their function. In 26 of the 30 cases the software predicted the class we expected. This means that in 86.6% of the cases, the ADENOLIM method lets us build sequences that belong to the binary class of our choice. We detail the results of the classification of the 30 samples and the backpropagation algorithm for derivative calculations in the Supplementary Information.

Finally, here we show the average values of certain features for the peptides having a therapeutic activity. The chosen features are the 4 most common amino acids (one-letter code), Hydrophobicity factor [34] (Hy), total negative charge (Tnc) and total positive charge (Tpc).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Antiviral | Tnc | Tpc | Hy | L | R | K | A |
| mean | -17.9 | 17.9 | -4.4 | 9.2% | 7.5% | 6.9% | 6.7% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Cell penetr. | Tnc | Tpc | Hy | R | K | L | A |
| mean | -17.7 | 17.7 | -4.4 | 18.7% | 13.1% | 9.4% | 7.2% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Antibac. | Tnc | Tpc | Hy | L | K | G | A |
| mean | -33.8 | 33.8 | -5.0 | 11.1% | 10.7% | 10.1% | 7.6% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Anticanc. | Tnc | Tpc | Hy | K | L | A | G |
| mean | -20.2 | 20.2 | -4.6 | 14.8% | 12.5% | 9.3% | 7.8% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Toxic | Tnc | Tpc | Hy | C | K | L | G |
| mean | -21.6 | 21.6 | -4.6 | 11.4% | 11.0% | 9.7% | 7.7% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Tumourh. | Tnc | Tpc | Hy | C | R | S | L |
| mean | -10.1 | 10.1 | -3.8 | 10.3% | 9.0% | 8.6% | 7.9% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Antihype. | Tnc | Tpc | Hy | P | L | V | G |
| mean | -7.1 | 7.1 | -3.7 | 14.9% | 9.2% | 7.5% | 6.4% |

We must not forget that these machine learning problems are not truly binary in nature. For example, different concentrations of the anticancer peptides would be needed to kill the same proportion of cancer cells. (cite: Marcelo paper, in press) Also, it is possible that a negative sequence will also possess the desired therapeutic activity at a very high concentration. Another example could be the cell penetrating case. Uptake efficiency [13] may be important in some applications and while it is always possible to estimate the efficiencies using the model-derived probabilities that a sequence is positive, taking the killing or uptake efficiencies into account and build a regression rather than a classification model is a much better way to achieve this goal. This is, however, particularly challenging due to the lack of data in the literature. Another thing worth mentioning is that gram-positive and gram-negative bacteria as well as haemolytic and cytotoxic proteins have been lumped together in the antibiotic and toxic case, respectively, and it is possible that separate prediction would have improved the results.

***Conclusion***. We developed a protocol for machine learning in proteomics and shared a user-friendly, easily-customizable and interpretable Python code using the Keras framework to make following the protocol easier. We demonstrated the usage of the pipeline on 7 different statistical learning examples and achieved satisfactory results in all cases. We described the 82 and a further 32 and 177 features that are extracted from the sequences by the pipeline and hypothesized that these features are good general descriptors of any small peptide sequence. Furthermore, we showed that using our pipeline makes statistical learning in proteomics faster, less laborious and yields result that are easier to interpret. We also came up with the ADENOLIM method, that lets us build sequences that belong to the binary class of our choice in 86.6% of the cases. For the user manual of the pipeline, please refer to the Supplementary Information and for the Python scripts, machine learning models, the datasets and the extracted vectors, see the GitHub project under the following link: ”https://github.com/sarvarip/Protein-prediction-pipeline”.

***References***.

[1] P. Vlieghe, V. Lisowski, J. Martinez, and M. Khrestchatisky, “Synthetic therapeutic peptides: science and market,” *Drug Discov. Today*, vol. 15, no. 1–2, pp. 40–56, 2010.

[2] G. Wang, “Human antimicrobial peptides and proteins,” *Pharmaceuticals*, vol. 7, no. 5, pp. 545–594, 2014.

[3] C. de la Fuente-Nunez, M. D. Torres, F. J. Mojica, and T. K. Lu, “Next-generation precision antimicrobials: towards personalized treatment of infectious diseases,” *Curr. Opin. Microbiol.*, vol. 37, pp. 95–102, 2017.

[4] S. Lata, N. K. Mishra, and G. P. S. Raghava, “AntiBP2: improved version of antibacterial peptide prediction.,” *BMC Bioinformatics*, vol. 11 Suppl 1, p. S19, 2010.

[5] A. Gautam *et al.*, “In silico approaches for designing highly effective cell penetrating peptides,” *J. Transl. Med.*, vol. 11, no. 1, p. 1, 2013.

[6] A. Sharma *et al.*, “Computational approach for designing tumor homing peptides.,” *Sci. Rep.*, vol. 3, no. Figure 1, p. 1607, 2013.

[7] W. Chen, H. Ding, P. Feng, H. Lin, and K. Chou, “iACP: a sequence-based tool for identifying anticancer peptides,” *Oncotarget*, vol. 7, no. 13, pp. 16895–16909, 2016.

[8] A. A. Bahar and D. Ren, “Antimicrobial peptides,” *Pharmaceuticals*, vol. 6, no. 12, pp. 1543–1575, 2013.

[9] S. Gupta, P. Kapoor, K. Chaudhary, A. Gautam, R. Kumar, and G. P. S. Raghava, “In Silico Approach for Predicting Toxicity of Peptides and Proteins,” *PLoS One*, vol. 8, no. 9, 2013.

[10] R. Kumar *et al.*, “An in silico platform for predicting, screening and designing of antihypertensive peptides,” *Sci. Rep.*, vol. 5, no. 1, p. 12512, 2015.

[11] A. Tyagi *et al.*, “CancerPPD: A database of anticancer peptides and proteins,” *Nucleic Acids Res.*, vol. 43, no. D1, pp. D837–D843, 2015.

[12] W. S. Sanders, C. I. Johnston, S. M. Bridges, S. C. Burgess, and K. O. Willeford, “Prediction of Cell Penetrating Peptides by Support Vector Machines,” *PLoS Comput. Biol.*, vol. 7, no. 7, 2011.

[13] M. Hansen, K. Kilk, and Ü. Langel, “Predicting cell-penetrating peptides,” *Adv. Drug Deliv. Rev.*, vol. 60, no. 4–5, pp. 572–579, 2008.

[14] Z. Hajisharifi, M. Piryaiee, M. Mohammad Beigi, M. Behbahani, and H. Mohabatkar, “Predicting anticancer peptides with Chou’s pseudo amino acid composition and investigating their mutagenicity via Ames test,” *J. Theor. Biol.*, vol. 341, pp. 34–40, 2014.

[15] K.-C. Chou, “Prediction of Protein Cellular Attributes Using Pseudo- Amino Acid Composition,” *Proteins Struct., Funct., Genet.*, vol. 255, no. September 2000, pp. 246–255, 2001.

[16] G. E. Hinton, S. Osindero, and Y. W. Teh, “A fast learning algorithm for deep belief nets.,” *Neural Comput.*, vol. 18, no. 7, pp. 1527–54, 2006.

[17] M. Hällbrink *et al.*, “Prediction of Cell-Penetrating Peptides,” *Int. J. Pept. Res. Ther.*, vol. 11, no. 4, pp. 249–259, 2005.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. 2016.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[20] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *Arxiv*, pp. 1–11, 2015.

[21] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *Conf. Pap. ICLR 2015*, pp. 1–15, 2015.

[22] A. Karpathy, “Convolutional Neural Networks for Visual Recognition,” *Stanford CS class CS231n notes*, 2017. [Online]. Available: http://cs231n.github.io/.

[23] S. Fritsch and F. Guenther, “neuralnet: Training of neural networks,” *R J.*, 2010.

[24] A. Ng, “Evaluating a Learning Algorithm,” *Coursera*, 2011. .

[25] J. Hastie, Trevor, Tibshirani, Robert, Friedman, *The Elements of Statistical Learning The Elements of Statistical Learning Data Mining, Inference, and Prediction, Second Edition*. 2009.

[26] F. Provost, “Machine learning from imbalanced data sets 101,” in *Proceedings of the AAAI’2000 workshop on imbalanced data sets*, 2000.

[27] D. S. Cao, Y. Z. Liang, J. Yan, G. S. Tan, Q. S. Xu, and S. Liu, “PyDPI: Freely available python package for chemoinformatics, bioinformatics, and chemogenomics studies,” *J. Chem. Inf. Model.*, vol. 53, no. 11, pp. 3086–3096, 2013.

[28] S. Singh *et al.*, “SATPdb: A database of structurally annotated therapeutic peptides,” *Nucleic Acids Res.*, vol. 44, no. D1, pp. D1119–D1126, 2015.

[29] P. Agrawal *et al.*, “CPPsite 2.0: A repository of experimentally validated cell-penetrating peptides,” *Nucleic Acids Res.*, vol. 44, no. D1, pp. D1098–D1103, 2016.

[30] D. A. Dobchev *et al.*, “Prediction of Cell-Penetrating Peptides Using Artificial Neural Networks.,” *Curr. Comput. Aided. Drug Des.*, 2010.

[31] A. Tyagi, P. Kapoor, R. Kumar, K. Chaudhary, A. Gautam, and G. P. S. Raghava, “In silico models for designing and discovering novel anticancer peptides.,” *Sci. Rep.*, vol. 3, p. 2984, 2013.

[32] I. Dbchak, I. Munchnik, S. R. Holbrook, and S.-H. Kim, “Prediction of protein folding class using global description of amino acid sequence,” *Proc. Natl. Acad. Sci. USA*, vol. 92, pp. 8700–8704, 1995.

[33] M. A. Nielsen, *Neural networks and deep learning*. Determination Press, 2015.

[34] R. Todeschini and V. Consonni, *Handbook of Molecular Descriptors*. Wiley, 2000.

***Supplementary information***.

1. Setting up the pipeline on Windows (user-friendly)

Follow the steps below to install Python:

* 1. Use this link: <https://conda.io/miniconda.html> to download Miniconda 64-bit version with Python 3.6
  2. After installation open the Anaconda Command Prompt and type the following to install and activate rdkit:
     1. Conda create -c rdkit -n my-rdkit-env rdkit
     2. Activate my-rdkit-env
  3. Install other required libraries by typing the followings into the Anaconda Command Prompt
     1. Conda install keras
     2. Conda install biopython
     3. Pip install pydpi
     4. Conda install scikit-learn
  4. In this case, we will be using the Theano backend with Keras
     1. First, make sure Theano is installed (may have already been installed by Keras) by typing: conda install theano
     2. Second, make sure that numpy is installed by typing: conda install numpy
     3. For Theano to be used, pygpu and libgpuarray need to be downgraded from default to 0.6.2, so type:
        1. conda install pygpu=0.6.2
        2. conda install libgpuarray=0.6.2
     4. Since Keras is based on TensorFlow by default, Theano needs to be set as the backend in the \_\_init\_\_.py file in the keras\backend folder. The exact path to this folder will vary, but as an example, my path is: C:\Users\User\Miniconda3\envs\my-rdkit-env\Lib\site-packages\keras\backend. This should be done manually, by editing this \_\_init\_\_.py document, for example, by Notepad, looking for backend and specifying ‘theano’.
  5. Replace all the pydpi .py files with the ones with the corresponding name from GitHub. The pydpi files will be found in the pydpi library in site-packages. The exact path to the folder will vary, but, as an example, my path is: C:\Users\Peter\AppData\Local\conda\conda\envs\my-rdkit-env\Lib\site-packages\pydpi. This ensures that the scripts are compatible with Python 3.6.
  6. Download an editor, for example, Atom (<https://atom.io/>). Download our protein\_prediction\_pipeline.py as well as inputoutput.py, config.py, modelbuilder\_pipeline.py and featurex\_pipeline\_v3.py scripts from GitHub. Save it into a new folder, which contains your positive and negative dataset in the “.fasta” or “.text” format. Then open the python script with your editor and edit the relevant fields (outlined in the code, see point 3).
  7. Copy the path of the folder you saved the script in and put cd (standing for change directory) in front of it in the Anaconda Command Prompt. In my case, this is: cd C:\Users\Peter\Desktop\MIT\code
  8. To start executing the script, type: python protein\_prediction\_pipeline.py
  9. If you close the Anaconda Command Prompt (or restart your computer), you need to do steps b/ii) and g again!

1. Setting up the pipeline on Google Cloud (advanced)
   1. Download the Google Cloud SDK Shell from: <https://cloud.google.com/sdk/downloads>
   2. Sign into your google account, go to: <https://console.cloud.google.com> and create a project and instance as described here: <http://cs231n.github.io/gce-tutorial/>
      1. Stick to the Ubuntu system, but increase the specifications if you are planning to use parallelization: we used 16 vCPUs and 14.4GB memory
   3. Take note of your instance name
   4. Open Google Cloud SDK and type: gcloud init
   5. Follow the instructions on screen and take note of the time zone you choose
   6. Create SSH by typing something like: gcloud compute ssh --zone=us-central1-c instance-2 , where you need to modify the zone (us-central1-c) and the instance name (instance-2) to your own time zone and instance name
   7. You should see the PuTTY window popping up
   8. Install Miniconda 3 and add it to your homedirectory by typing
      1. wget [http://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86\_64.sh -O ~/miniconda.sh](http://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh%20-O%20~/miniconda.sh)
      2. bash ~/miniconda.sh -b -p $HOME/miniconda
      3. export PATH="$HOME/miniconda/bin:$PATH"
   9. Install required things by typing words in *italics*
      1. Install rdkit (same as under poin 1): *conda create -c rdkit -n my-rdkit-env rdkit*
      2. Activate environment: *source activate my-rdkit-env*
      3. *Conda install keras* (tensorflow should automatically come with it, luckily on linux it is compatible with rdkit!)
      4. *Conda install biopython*
      5. *Pip install pydpi*
      6. *Conda install jupyter notebook*
      7. *Conda install sickit-learn*
      8. Use the cs231n description (<http://cs231n.github.io/gce-tutorial/>) to make static IP and set firewall rule on konzol, then use nano to append to the end of the juypter init file and use paste to create the ‘\*’ (I could not figure out any other way other way to type a \* in nano other than pasting it from clipboard)
      9. *jupyter-notebook --no-browser --port=7000*
      10. Navigate to the required static IP ‘/’ <Port number> (for example, <http://164.178.50.189:7000> ) by typing it into your browser and use token given in the PuTTy command window in the hyperlink that it asks you to paste into browser (DO NOT PASTE THAT THOUGH, just use the token from the link, after ‘token=’) To find IP, see external IP for the VM instance you are using on the google cloud console.
   10. Upload the required files (protein\_prediction\_pipeline.py as well as inputoutput.py, config.py, modelbuilder\_pipeline.py and featurex\_pipeline\_v3.py scripts from GitHub and your positive and negative datasets) using the upload tab on the browser after opening jupyter notebook and edit the relevant fields of the Python script (see point 3)
   11. Since Python version 3 is used, we need to overwrite the print and include commands in the pydpi in the following path (your path will be the same, but the second directory will be your name rather than mine): "/home/Peter/miniconda/lib/python3.6/site-packages/pydpi " in folders drug as well as protein and “pydrug.py”. Since we have done this already, just replace the imported files in the relevant directories with the ones we uploaded to GitHub.
   12. Use CTRL-C in the PuTTY window to terminate Jupyter Notebook and type: python protein\_prediction\_pipeline.py to execute the script
   13. If you get an error (I did) while loading shared libraries, here is the solution: <https://www.cyberciti.biz/faq/debian-ubuntu-linux-wkhtmltopdf-error-while-loading-shared-libraries-libxrender-so-1/>

Basically, search for the missing libs using “apt-cache search libXrender”, which returns the missing libraries and then install them using, for example, “sudo apt-get install libfontconfig1 libxrender1”

* 1. If you close your PuTTY window (the one that appeared after you computed the ssh in step f) you need to do step f), h/iii) and i/ii) again! If you restart your computer you will also need to open Google Cloud SDK first. If you want to use Jupyter notebook you will need to do steps i/ix) and i/x) again.

1. User manual for editing the protein\_prediction\_pipeline.py script

After opening the script, look for this well outlined portion (here I have deleted the comments to make this more readable):

'''User interface'''

'''Options start'''

vector\_name = "pipeline\_default"

model\_name = "pipeline\_default"

numcores = 4

choice = 0

reduce\_by\_similarity = 1

use\_random\_small\_sequence\_negative = 0

already\_extracted = 0

pos\_input\_name = "insert\_name"

neg\_input\_name = "insert\_name"

'''Only If prediction (choice 6) is chosen'''

predict\_pos\_input\_name = "insert\_name"

predict\_neg\_input\_name = "insert\_name"

known\_classes = 1

'''Only change if you get error regarding the sampling or if you are notified that your dataset s not balanced and asked to increase negative dataset sampling'''

sc\_1 = 2

sc\_2 = 1.25

'''Options end'''

'''User interface'''

* 1. Vector\_name refers to the name of the pickle file in which you store the extracted features of your sequences. For the positive dataset the name of the pickle file will be the vector name preceded by “pos\_” and for the negative dataset it will be the vector name preceded by “neg\_”.
  2. Model\_name refers to the names of the files that save the data associated with the trained model. This data is the model structure (json file), the weights data (h5 file), the mean and the standard deviation used for scaling the input data (preceded by “mean\_” and “std\_” respectively, pickle files). I recommend to make the vector name and model name the same, for example, if you are building a classifier with sequences having anti-parasitic effect, you could name it “pipeline\_antiparasitic”.
  3. The numcores variable refers to the number of cores that your machine has. It is set to 4 by default, change it according to the number of processors you have. Only available in protein\_prediction\_pipeline\_parallel.py.
  4. For choice, make it equal to the number that describes the option you would like to use (see Methods section). Default is zero meaning 10-folds cross-validation.
  5. Reduce\_by\_similarity should be one, unless you have a specific negative dataset (see Methods section).
  6. Use random\_small\_sequence\_negative should be zero, unless, you would like to use the sample negative dataset provided by us. If so, you will need to download this from GitHub. This sequence is the negative dataset we used for the anti-cancer classification (see description in Data Mining section).
  7. Already\_extracted should be set to one, if you are have already run the script with these inputs (datasets) before, but now you would like to change the model (evaluation: set choice to a different number or -advanced- use a custom-written modelbuilder.py). This is to save time, since the features have already been extracted, so there is no need to do that again.
  8. Pos\_input\_name and neg\_input\_name is the name of the positive and negative fasta or txt files that contain the positive sequences (with a specific activity) and the negative sequences, respectively. For example, in the antibacterial case my neg\_input\_name is "uniprot\_3\_100\_noannotationantibacterial.fasta".
  9. Predict\_pos\_input\_name and predict\_neg\_input\_name should not be edited unless you use choice = 6 (prediction), in which case you will use one of your existing models to predict the classes of a new set of data. If you know which class the sequences in this new data belong to, put the positive sequences in a text/fasta file and make the name of the file equal to predict\_pos\_input\_name. Then put the negative sequences in another file and assign it to predict\_neg\_input\_name. If you have no such information, simply partition the sequences into two files (does not matter how you do it and they do not need to contain the same amount of sequences) and set known\_classes to zero. The probabilities you will see correspond the order of sequences in the negative dataset and then that of the positive dataset.
  10. Sc\_1 and sc\_2 and the sampling coefficients. You should only change them if you get an error message regarding the sampling or if you are asked to do so, because the classes are imbalanced. In the first case (sampling error), you should increase sc\_1 slightly (e.g. to 3 – take care than in this case the negative dataset size should be about 3 times the size of the positive dataset!) In the class imbalance case, you should increase sc\_2 (e.g. to 2). Sc\_2 should always remain smaller than sc\_1.

Examples about the usage of the options can be seen on the Github: the parallel versions of the scripts used to predict the 7 therapeutic activities can be accessed via <https://github.com/sarvarip/Protein-prediction-pipeline/tree/master/examples>.

1. FAQ:
   1. Problem: I got out-of-bound error when the script was trying the access the elements of an array.

Solution: Make sure that your negative dataset contains at least two times as many samples as the positive one. If not, reduce sc\_1, but make sure it is bigger than sc\_2.

* 1. Question: Which featurex online code shall I use?

Answer: It depends.

* + 1. If you would like to use the 82 features, use featurex\_pipeline\_v3.py.
    2. If you would also like to use the Moreau-Broto autocorrelation, use featurex\_pipeline.py. In this case you need to download this file from GitHub too, put it in the same folder as protein\_prediction\_pipeline.py is and rewrite the import in line 13 to “import featurex\_pipeline as FX”.
    3. If you would like to use the 281 features, use featurex\_pipeline\_v3\_pseaac\_aac.py. In this case you need to download this file from GitHub too, put it in the same folder as protein\_prediction\_pipeline.py is and rewrite the import in line 13 to “import featurex\_pipeline\_v3\_pseaac\_aac as FX”.
    4. If you would like to get a quick baseline prediction with the amino acid composition features (20), use featurex\_pipeline\_aac.py. In this case you need to download this file from GitHub too, put it in the same folder as protein\_prediction\_pipeline.py is and rewrite the import in line 13 to “import featurex\_pipeline\_aac as FX”.
    5. Advanced: Using the featurex scripts as templates, and reading the pydpi paper, you can easily customize feature extraction.

1. Tumour homing and antiviral prediction results using the default 82 features

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Prediction according to ML protocol | Matthew’s Correlation Coefficient (MCC) | Accuracy (%) | F1 score | Precision | Recall |
| Antiviral | 0.58(+/-0.06) | 79.49(+/-2.25) | 0.78(+/-0.05) | 0.79(+/-0.04) | 0.79(+/-0.03) |
| Cell penetrating | 0.77(+/-0.04) | 89.89(+/-1.48) | 0.87(+/-0.04) | 0.88(+/-0.02) | 0.89(+/-0.02) |
| Antibacterial | 0.73(+/-0.03) | 86.23(+/-1.82) | 0.87(+/-0.02) | 0.87(+/-0.02) | 0.87(+/-0.02) |
| Anticancer | 0.60(+/-0.04) | 80.88(+/-3.84) | 0.78(+/-0.04) | 0.80(+/-0.02) | 0.80(+/-0.02) |
| Toxic | 0.72(+/-0.02) | 86.72(+/-1.55) | 0.85(+/-0.02) | 0.86(+/-0.01) | 0.86(+/-0.01) |
| Tumour homing | 0.69(+/-0.05) | 83.79(+/-3.97) | 0.85(+/-0.04) | 0.84(+/-0.02) | 0.84(+/-0.02) |
| Antihypertensive | 0.75(+/-0.05) | 87.99(+/-3.29) | 0.87(+/-0.05) | 0.87(+/-0.03) | 0.87(+/-0.02) |

1. Tumour homing and antiviral prediction results using the 281 features

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Prediction according to ML protocol | Matthew’s Correlation Coefficient (MCC) | Accuracy (%) | F1 score | Precision | Recall |
| Antiviral | 0.61(+/-0.04) | 80.45(+/-2.13) | 0.81(+/-0.04) | 0.81(+/-0.03) | 0.81(+/-0.02) |
| Cell penetrating | 0.77(+/-0.04) | 87.96(+/-4.08) | 0.89(+/-0.05) | 0.88(+/-0.02) | 0.88(+/-0.02) |
| Antibacterial | 0.76(+/-0.02) | 88.00(+/-1.37) | 0.87(+/-0.01) | 0.88(+/-0.01) | 0.88(+/-0.01) |
| Anticancer | 0.63(+/-0.05) | 81.56(+/-3.89) | 0.81(+/-0.05) | 0.81(+/-0.03) | 0.81(+/-0.03) |
| Toxic | 0.72(+/-0.04) | 85.64(+/-1.72) | 0.87(+/-0.03) | 0.86(+/-0.02) | 0.86(+/-0.02) |
| Tumour homing | 0.70(+/-0.07) | 85.58(+/-3.76) | 0.84(+/-0.06) | 0.85(+/-0.04) | 0.85(+/-0.04) |
| Antihypertensive | 0.78(+/-0.05) | 89.27(+/-2.70) | 0.89(+/-0.04) | 0.89(+/-0.03) | 0.89(+/-0.03) |

1. ADENOLIM Method
   1. Classification of the samples

|  |  |  |
| --- | --- | --- |
| Sequence | Expected | Software prediction |
| GFCK | + | + |
| CCGKKK | + | + |
| CGGCF | + | + |
| FFKKKF | + | - |
| KKCGCCGG | + | + |
| GGGG | + | - |
| GFKCK | + | - |
| CCCFFC | + | + |
| FKCKKK | + | + |
| GFGFF | + | + |
| GFGKCGCK | + | + |
| FFCKGG | + | + |
| CCKGG | + | + |
| GCKKFFCCFKGCF | + | + |
| KKGFFKG | + | + |
| MDETPVQ | - | - |
| DQMEDTT | - | - |
| EEQMMVV | - | - |
| MMMMEM | - | - |
| MMDPT | - | - |
| VPPVTQE | - | - |
| DVDVDEQ | - | - |
| EMMEMM | - | + |
| QTVTQD | - | - |
| EEETTEE | - | - |
| EQDPPPE | - | - |
| DPDEVMQPDMTME | - | - |
| EQTPDM | - | - |
| DQMMTTT | - | - |
| VEDPVVTE | - | - |

MCC = 0.74, F1-score = 0.87

To replicate the results go to: <https://github.com/sarvarip/Protein-prediction-pipeline/tree/master/ADENOLIM> download all the files (except for the .csv files, which reflect the results), run ADENOLIM.py to have a look at the ADENOLIM values and run pipeline\_complete\_antibac\_choice6.py to see the classification results of the sample sequences (see above) designed using the ADENOLIM method.

* 1. Investigation whether the slopes of a feature have the same sign in all cases or not:

The easiest way to deduce this is compare two results: one is the absolute value of the sum of the slopes and the other is the sum of the absolute values of the slopes. If the two are equal for a feature, then it must mean that the derivative of the output with respect to that feature has the same sign in all cases (through all samples).

* 1. Forward – and backpropagation [33] and also see: http://cs231n.github.io/optimization-2/

To make the backpropagation easier, we removed batch normalization from the model when using the ADENOLIM method. We use the ReLU nonlinearity between the layers. We have 4 weights matrices:

* + 1. W1, whose dimension is input features \* number of neurons in the first hidden layer = 20\*64
    2. W2, whose dimension is 64\*32
    3. W3, whose dimension is 32\*16
    4. W4, whose dimension is 16\*1

The input vector has dimension 1\*20 and we also have 4 biases, b1, b2, b3 and b4.

Hence the equations for the forward propagation are

Equations for the backpropagation:

The square brackets in the equations mean that wherever the elements in the matrix in the square bracket fulfil the inequality, the assignment occurs for the element occupying the same places in the leftmost matrix.